Generalizing Pre-Trained Neural Language Models

Examen de doctorat

Nicolas Garneau

Doctorat en Informatique
Philosophiæ doctor (Ph. D.)

Québec, Canada

Résumé

Dans sa forme la plus générique, un modèle de langue est entraîné à prédire le prochain symbole étant donnée un historique de symboles. Les symboles peuvent être des *mots* ou des *caractères* par exemple. Entraîné sur un corpus de texte, un modèle de langue peut donc prédire la probabilité d'une séquence de symboles de longueur indéterminée. Les modèles de langue sont utilisés dans plusieurs applications du traitement automatique de la langue naturelle telle que la traduction automatique où l'on s'en sert pour obtenir la traduction la plus probable dans la langue cible étant donné une liste de candidats. Dans les dernières années, les réseaux de neurones sont devenus l'état de l'art en matière de modélisation de la langue. Il a été démontré que ces modèles de langue préentraînés sont intrinsèquement des extracteurs de caractéristiques (représentations langagières) avec lesquels on peut ajouter un simple classificateur pour accomplir une tâche spécifique comme de la classification de texte ou de l'étiquetage de séquence. Le web contient un volume de données textuelles colossal et nous voyons maintenant l'apogée de modèles plus imposants en nombre de paramètres qui poussent les limites de l'état de l'art encore plus loin. Cependant, ces puissants modèles de langue préentraînés peuvent seulement être transférés efficacement vers des tâches de la même langue qui est limitée principalement à l'anglais pour le moment.

Même si ces modèles obtiennent des performances exceptionnelles sur une langue spécifique, les transférer vers une autre langue est une avenue de recherche sous-explorée. Dans cette proposition, nous présumons que d'utiliser un modèle de langue préentraîné sur un corpus anglophone profitera tout autre modèle de langue tout comme aux tâches que nous tentons d'accomplir dans la langue cible qui en découlent (e.g étiquetage de séquences en danois). L'objectif principal de cette proposition est d'explorer le transfert multilingue des modèles de langue préentraînés, donc de les *généraliser* à des langues sous représentées en termes de ressources textuelles. Pour atteindre cet objectif, nous nous inspirons du travail qui a été accompli pour transférer des représentations de mots *fixes*. Ce champ de recherche actif a généré plusieurs méthodes pour transférer de telles représentations préentraînées ce qui constitue le point de départ de nos expérimentations.

Transférer des représentations langagières vers des tâches de même langue assume intrinsèquement un vocabulaire similaire entre le modèle de langue et le jeu de données de la tâche. Transférer un modèle de langue préentraîné d'une langue source vers une langue cible invalide cette hypothèse, surtout si les deux langues ou jeux de données ont un lexique très différent.

Nous prévoyons également explorer l'utilisation de métadonnées externes tels que des images ou le profil d'un usager pour conditionner un modèle de langue préentraîné dans un contexte conversationnel où le champ lexical diffère du modèle pré-entraîné.

Pour terminer, nous proposons d'étudier les dynamiques d'apprentissage des modèles de langue dans un contexte multilingue en contraignant les représentations vectorielles à l'aide d'un jeu de données composé d'analogies.

Le résultat de ce projet sera de fournir diverses manières de transférer des représentations de mots *contextuelles* obtenues à l'aide d'un modèle de langue neuronal anglophone vers une langue ou un champ lexical cible.

Abstract

In its most generic form, a language model is trained to predict the next symbol given a history of symbols. Symbols can either be *words* or *characters* for example. Trained on a corpus, a language model can therefore predict the probability of a sequence of symbols of indefinite length. The language models are used in several applications of natural language processing such as machine translation where it is used to obtain the most likely translation into the target language given a list of candidates. In recent years, neural networks have become the state of the art in language modelling. It has specifically been shown that these pre-trained language models are inherently feature extractors (language representations) with which one can add a simple classifier to accomplish a specific task like text classification or sequence labelling. The Web contains a colossal amount of textual data and we now see the apogee of more imposing models in number of parameters which push the state of the art even further. However, these powerful pre-trained language models can only be effectively transferred to tasks of the same language which is limited mainly to English for the moment.

While these models obtain exceptional performances on a specific language, transferring them to another language is an under-explored research avenue. We assume that using an English pre-trained language model will benefit to any other language model as well as the downstream tasks in the target language (e.g. sequence labelling in Danish). The main objective of this proposal is to explore the multilingual transfer of pre-trained language models, therefore to *generalize* the models to languages which are under-represented in terms of textual resources.

Transferring language representations from one task to another inherently assume a similar vocabulary between the language model and downstream task's dataset. Transferring a pre-trained language from a source language to a target language invalidates this hypothesis, especially if the two languages or the dataset have a very different lexicon.

To achieve our goal, we inspire ourselves by the work that has been accomplished to transfer *fixed* word representations from a source to a target language. This active field of research has generated several methods to transfer such pre-trained word representations which is the starting point for our experiments.

We also plan to explore the use of external metadata such as images or user profiles to condition a pre-trained language model in a conversational context where the lexicon is different from the pre-trained model's.

Finally, we propose to study the learning dynamics of pre-trained language models in a multi-

lingual setting by grounding the vectorial representations using a multilingual analogy dataset.

The result of this project will be to provide a way of transfer *contextual* representations of words to a target language obtained using an English pre-trained neural language model.

Contents

List of Figures

To my wife Laurie-Anne who believes in me and my bouncing ideas.

Enjoy your problems.

Shunryu Suzuki

Remerciements

Je remercie Luc Lamontagne, mon directeur de recherche depuis maintenant 6 ans qui m'a donné le goût de faire de la recherche. Il a toujours été présent pour me conseiller à propose de mon parcours académique, professionnel et personnel.

Je remercie également mes collègues du laboratoire de recherche en apprentissage automatique de Laval, sans qui les journées n'auraient pas été remplies de moments de réflexion et de bons fou rires. Être sérieux dans notre recherche est important pour bien performer mais du plaisir l'est d'autant plus.

Introduction

In the past few years, the natural language processing (NLP) area has received a considerable amount of attention from researchers in the community of machine learning. The resurgence of neural networks is undoubtedly for something. We have seen multiple neural architectures outperforming classical methods in a wide range of natural language understanding tasks. A key component that lies within these architectures are language models.

Language models are traditionally known as having the capacity of evaluating the probability of a given textual utterance as well as generating text. Such a feature becomes handy in many natural language understanding applications. It is used in machine translation to obtain the most probable translation given a list of potential candidates in the target language, for example.

Modelling a language is a hard challenge. Every language hides its subtle complexities that make them ambiguous, thus hard for a machine to reason about. However, several languages share syntactic and semantic commonalities. Leveraging these similarities to improve a punctual model on a specific language is thus a valid assumption. Hence, cross-lingual learning is thoroughly studied, especially with the rise of neural networks where knowledge transfer is a common practice.

A language model is often used in a specific context. Other than some linguistics aspects that may condition a language model like the previous word in a conversation, many other elements can be considered in order to better condition such a model. Image captioning is one of the first applications of conditional language modelling. It is possible to broaden the context with a user's profile, whole documents, business rules, etc. in order to specialize a given language model.

In this research proposal, we analyze the structure and the usability of neural language models. With the outstanding amount of textual data available on the web, we have seen the rise of new language model architectures able to ingest and distill a large amount of linguistic information. These recent architectures pushed forward the state of the art in many natural language understanding applications but mainly for one language: English. Even though we begin to see the publication of these models in a few other languages, they require a considerable amount of data and computational power to be properly trained. This leaves underrepresented languages in the dark and only wealthy companies can afford to train from scratch these architectures.

We thus propose to study the cross-lingual transfer of these pre-trained English models, hence to generalize these models towards lower-resource languages and alleviate the need for a huge amount of textual data and computational resources. We also propose to broaden the contextual conditioning of such architectures that strictly handle textual conditioning. Still, the common thread of this research proposal is to leverage and analyze pre-trained language models as black boxes. We do not aim to improve these architectures directly, but rather their use in a specific context.

The first chapter of this proposal introduces the basis of statistical language modelling. It also proposes a brief overview of neural networks and the primary architectures and components that are commonly used in natural language processing such as word embeddings.

The second chapter provides an extensive overview of the different language model architectures, evaluation schemes and training procedures that are specific to neural networks. It also provides a thorough analysis of the neural language generation in a free-form manner as well as conditioned generation.

Chapter three introduces the first two research avenues that specifically tackle the two issues previously introduced. The very first avenue is about cross-lingual neural language transfer. We propose several steps that we hope will lead us to the ability to transfer a source language model towards a target language with minimal resources and effort. The second avenue is framed along with the evaluation of language models. We propose to apply a cross-lingual neural model on a question answering dataset that induces several contextual elements such as PDF forms and user profiles. We hope that these contextual elements will be of great use for the language model to generate specialized textual content. The last research avenue aims to study the learning dynamics of a pre-trained multilingual language model using a specially curated cross-lingual analogy dataset.

Chapter 1

Background

1.1 N-Gram Language Models

In this section, we introduce the generic mathematical notation of a language model and how we can practically model language using a traditional method called N-Grams.

1.1.1 Language Models

In the most generic form, language models are trained to predict the next symbol s_i given a sequence of k previous symbols $(s_{i-k}, \ldots, s_{i-1})$ using a textual corpus. More specifically, language models are designed to estimate the following conditional probability;

$$P(s_i|s_{i-k}, \ldots, s_{i-1}). \tag{1.1}$$

Symbols can either be *words*, *characters* or *byte-pairs*, which we detail in section 2.1. This makes language models inherently **decoders** or **generators**, being able to generate text given a history of symbols. The generation of symbols can either be greedy or optimized (which we further discuss in Section B.0.1).

Subsequently, language models can predict the probability of a given sequence S of k symbols;

$$P(S) = P(s_1, \ldots, s_k). \tag{1.2}$$

A commonly used metric derived from this probability is the *perplexity*, which we further describe in Section 2.2.

In the next section, we present a method that has been used for years to model language and is still a strong baseline, the N-Gram language model.

1.1.2 N-Gram Language Models

It is possible to compute **Equation 1.2** for a given sequence S of length k using the chain rule of probability;

$$P(S) = P(s_1) \cdot P(s_2|s_1) \cdot P(s_3|s_1, s_2) \cdot \ldots \cdot P(s_k|s_1, s_2, \ldots, s_{k-1}). \tag{1.3}$$

In practice, we approximate **Equation 1.3** using a sequence of n-grams of symbols, restricting the effective history of each probability to be of $n-1$ symbols. For a bigram language model, for example, we approximate the probability of a given symbol based only on the preceding symbol $P(s_i|s_{i-1})$.

Using a similar notation as in Jurafsky and Martin (2018), we can define the probability of a given sequence S of length k using the bigram assumption as follows:

$$P(s_i^k) \approx \prod_{i=1}^{k} P(s_i|s_{i-1}). \tag{1.4}$$

We can generalize Equation 1.4 to the n-gram case;

$$P(s_i^k) \approx \prod_{i=1}^{k} P(s_i|s_{i-n}, \ldots, s_{i-1}). \tag{1.5}$$

We obtain such probabilities of n-grams using the maximum likelihood estimation (MLE) by computing the occurrence counts in a corpus. In the bigram case, we can obtain the probability of a symbol s_i given the preceding one s_{i-1}:

$$P(s_i|s_{i-1}) = \frac{C(s_{i-1}, s_i)}{C(s_{i-1})} \tag{1.6}$$

where $C(s_{i-1}, s_i)$ is the number of times the symbol s_{i-1} appears *before* the symbol s_i in the corpus and $C(s_{i-1})$ is the number of times s_{i-1} appears in the corpus. One can thus, having a proper corpus in hand and a n-gram language model, predict the next symbol (i.e. *word*) of the following sentence:

```
Nicolas walks towards the bar to order a glass of ...
```

Given the context, words that first come up to mind might be `water`, `beer` or `wine`. Those that personally know Nicolas would assign the word `whisky` a high probability of being generated![1]

Usual n-gram models will use bigrams, trigrams and even up to 5-grams of history to model a particular language. It goes without saying that a bigger history needs a bigger corpus to train such n-gram models. Google computed the counts for up to 5-grams on a training corpus

[1] As you can see, this would be a perfect case for conditional language modelling, leveraging the taste profile of Nicolas in order to generate the most probable word according to his favourite drinks.

of one trillion words from public Web pages. They released the data via the Linguistic Data Consortium (LDC)[2]. n-gram language model limitations are further discussed in Section 2.1.2.

Neural networks are going to be the main tool to conduct the experiments during this research. They are, in many natural language processing applications, the state-of-the-art models so it is worth mentioning the main varieties of neural architectures used in language modelling.

We begin by introducing Word Embeddings which are an essential building block for representing words as vectors. We also present Cross-Lingual Word Embedding learning since we are going to make use of these models in Chapter 3. We then present the feed-forward neural architecture followed by the recurrent neural networks. We conclude this section with the recent Transformer architecture which pushed the state-of-the-art at an unprecedented level in many NLP tasks.

1.2 Word Embeddings

Words are categorical values that need to be converted into vectors in order to be used within a machine learning model. One method of vectorizing words is one-hot encoding. Given a matrix $\mathbf{E} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$, each line $i = 1, \ldots, |V|$ is represented by a perpendicular vector $\mathbf{w}^i$ where

$$\mathbf{w}^i_i = 1 \text{ and } 0 \text{ elsewhere.} \tag{1.7}$$

We define a lookup function $h(s) = i$ that takes as input the symbol s and returns its index i in the embedding table $\mathbf{E}$. Thus, the vector representation of the specific word "cat" is obtained the following way;

$$\mathbf{E}[h(\texttt{cat}),] = \mathbf{w}_{\texttt{cat}} = [\underset{\texttt{dog}}{0}, \underset{\texttt{paw}}{0}, \ldots, \underset{\textbf{cat}}{\mathbf{1}}, \ldots, \underset{\texttt{milk}}{0}, \underset{\texttt{hat}}{0}] \tag{1.8}$$

where $\mathbf{w}_{\texttt{cat}} \in \{0,1\}^{|\mathcal{V}|}$ is the row vector representation for the word cat. We can see $\mathbf{E}$ as a simple lookup matrix.

One-hot encodings have major limitations. For example, using vector arithmetic such as the cosine distance between one vector and another always yields zero. Being able to perform vector arithmetic between words is an important feature that may facilitate any downstream natural language processing tasks.

Take for example the words "cat" and "kitten" that have similar meanings. Having similar word vectors for these words would be beneficial when comes the time to assign them a particular attribute such as in the task of part-of-speech tagging (POS) for example.

We also use the cosine distance as a measure of how two words are far apart, making one-hot encoding unsuitable for this metric;

[2]`https://catalog.ldc.upenn.edu/LDC2006T13`

$$\text{cosine\_distance}(\mathbf{w}_1, \mathbf{w}_2) = 1 - \frac{\mathbf{w}_1 \cdot \mathbf{w}_2}{||\mathbf{w}_1|| \cdot ||\mathbf{w}_2||}. \tag{1.9}$$

A more judicious way to represent words is by using contextual vectors. Again, a word vector $\mathbf{w} \in \mathbb{N}^{|\mathcal{V}|}$ is used but instead of using the one-hot representation, we count every time another word appear within the context [3] of that target word and assign this count to the index of this specific context word;

$$\mathbf{E}[h(\texttt{cat}),] = \mathbf{w}_{\texttt{cat}} = [\underset{\textbf{dog}}{\mathbf{1}}, \underset{\textbf{paw}}{\mathbf{4}}, \ldots, \underset{\texttt{cat}}{0}, \ldots, \underset{\textbf{milk}}{\mathbf{3}}, \underset{\texttt{hat}}{0}] \tag{1.10}$$

Working with these vectors can be prohibitive if the vocabulary size is very large. We can reduce the dimensionality of these words representations by performing Singular Value Decomposition (Wall et al.), keeping only the top k singular values. We can thus obtain a matrix of word embeddings $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times k}$ instead of $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$. k values range from a few hundred to a thousand.

Recent advances in learning dense representations for words are using shallow neural networks. Mikolov et al. proposed 2 architectures, *Skip-Gram* and *CBOW* parameterized by two embedding matrices, $\Theta = \mathbf{E}^t \in \mathbb{R}^{|\mathcal{V}| \times k}, \mathbf{E}^c \in \mathbb{R}^{|\mathcal{V}| \times k}$, which are the target and context embeddings respectively. At the end of the training loop, only $\mathbf{E}^t$ is used as the embedding matrix for the vocabulary $\mathcal{V}$. The first method (*Skip-Gram*) tries to predict context words that appear within a specific window of a target word. *CBOW* on the contrary tries to predict the target word given its context.

Given a sequence of training words $w_1, w_2, w_3, \ldots, w_N$, the objective of the *Skip-Gram* model is to maximize the average log probability of the following objective function;

$$\frac{1}{N} \sum_{n=1}^{N} \sum_{-i \leq n \leq +i, i \neq 0} \log\, p(w_{n+i} | w_n) \tag{1.11}$$

where i is the context window size. A wise trick that alleviates the prohibitive computation of the previous conditional probability is negative sampling. Instead of computing the probability over the whole vocabulary $\mathcal{V}$, this method only uses k negative samples. Mathematically, the objective function using negative sampling is defined as the following loss function with respect to the parameters Θ;

$$\mathcal{J}(\Theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{-i \leq n \leq +i, i \neq 0} \log\, \sigma(\mathbf{w}_i^{c\top} \mathbf{w}_n^t) + \sum_{j=1}^{k} \log\, \sigma(-\mathbf{w}_j^{c\top} \mathbf{w}_n^t) \tag{1.12}$$

where k is the number of negative samples randomly selected from the vocabulary. The CBOW model can similarly be express in terms of negative sampling and has thus been omitted.

[3] The context is usually defined by a window of words that surround the target word. Typical window sizes span from 1 to 10 words on each side of the target word.

Pennington et al. on the other hand uses a similar architecture to predict the magnitude of co-occurrence of a target word and a word that appeared in its context (also based on a window size) by adding two sets of biases parameters $\mathbf{b}^t \in \mathbb{R}^{|\mathcal{V}|}$ and $\mathbf{b}^c \in \mathbb{R}^{|\mathcal{V}|}$. The model is defined with the following loss function;

$$\mathcal{J}(\Theta) = f(x_{i,j})(\sum_{i,j=1}^{|\mathcal{V}|} \mathbf{w}_i^{t\top} \mathbf{w}_j^c + b_i^t + b_j^c - \log x_{i,j})^2 \tag{1.13}$$

where $x_{t,c}$ is the number of times the word w_c appeared in the context of the word w_t and $f(\cdot)$ is a weighting function that discounts rare and frequent co-occurrences. The previously cited methods specifically tackled the word level embeddings but they also generalize to characters and BPE levels.

A common evaluation scheme of such word embedding models is via the analogical reasoning task introduced by Mikolov et al.. The task consists of finding analogies such that Berlin is to Germany what **Paris** is to France using simple vector arithmetic;

$$\mathbf{w}_{\text{Berlin}} - \mathbf{w}_{\text{Germany}} + \mathbf{w}_{\text{France}} \approx \mathbf{w}_{\text{Paris}} \tag{1.14}$$

We further detail how we are going to use this specific task in Section 3.3.

1.2.1 Cross-Lingual Word Embedding Models

The previously introduced word embedding models have been extensively used in a monolingual setting, consistently improving natural language understanding models in every downstream task. However, these downstream tasks usually need a considerable amount of annotated data in order to be effective, which leaves underrepresented language apart. This raised the need to transfer word embeddings trained in a *source language* (e.g. English) with much more training data towards a *target language* (e.g. Danish) that has much less annotated resources.

Ruder et al. provide a good overview of the different methods that are available to date for transferring word embeddings from one source language to a target language. They suggest three different types of **word alignment models** and in this proposal, we will introduce only the mapping-based approach since it is the most popular and is fulfilled with many annotated resources. We refer the reader to Ruder et al. (2018) for details on other types of alignment. The goal of word alignment is to find a joint cross-lingual embedding space where the source and target word embeddings are projected.

Mapping method. There are actually four different mapping methods (regression, orthogonal, canonical and margin). We plan to use the regression method since it perfectly suits our case where one source embedding space may not be in the same dimension as the target embedding space (which is required for more robust orthogonal methods). A regression mapping learns a linear projection with a transformation matrix $\mathbf{W}^{s\rightarrow t}$ that maps the vector space of the source language s to the vector space of the target language t. $\mathbf{W}$ is learned by minimizing the following mean squared error (MSE) loss;

$$\mathcal{J}_{MSE}(\mathbf{W}) = \sum_{i=1}^{n} ||\mathbf{W}\mathbf{x}_i^s - \mathbf{x}_i^t||^2 \tag{1.15}$$

where $\mathbf{x}_i^s$ and $\mathbf{x}_i^t$ are the i^{th} source and translated target word embeddings respectively. This assumes that we have a **seed lexicon** in hand (often called the seed dictionary) that is used to obtain translation pairs from a particular word w^s from the source language in the target language w^t.

Seed lexicon. For our experiments, we plan to use an off-the-shelf bilingual lexicon that has been learned using an internal translation tool at Facebook[4]. This dataset comprises 110 large-scale ground-truth bilingual dictionaries with up to 100,000 pairs of translations. These quality lexicon are more than sufficient to train a mapping model since it requires at least 5000 pairs (Ruder et al., 2018).

Refinement. It has been shown (Hartmann et al., 2019) that mapping-based approaches (either supervised or unsupervised) usually need to refine the mapping between the source and target pairs (Artetxe et al., 2018). This problem arises when the seed lexicon is composed of less than 5000 pairs (Vulic and Korhonen, 2016). Since we'll be using many translation pairs as previously stated, we won't be tackling this issue at first but it may prove to be necessary after preliminary experiments.

Retrieval. Cross-lingual embedding models are often used to **retrieve** the translation of a specific source word given its embedding. Suppose that we have the word "cat" and we want to retrieve its French translation. One of the possible translations could be "chat", "chaton" or even "félin". To do so, we'll use the "cat" word embedding $\mathbf{x}_{\mathtt{cat}}^{\mathtt{e}}$ taken from the English embedding table $\mathbf{E}^e$ and the pre-trained projecting matrix $\mathbf{W}^{\mathtt{e}\to\mathtt{f}}$ to obtain the projected "cat" representation in the French target language;

$$\hat{\mathbf{x}}_{\mathtt{cat}}^{\mathtt{f}} = \mathbf{W}^{\mathtt{e}\to\mathtt{f}}\mathbf{x}_{\mathtt{cat}}^{\mathtt{e}} \tag{1.16}$$

where $\hat{\mathbf{x}}_{\mathtt{cat}}^{\mathtt{f}}$ constitutes our best estimated representation of the word "cat" in French. We then use the pre-trained French embedding matrix $\mathbf{E}^{\mathtt{f}}$ to retrieve the most similar word from the vocabulary;

$$h(w^f) = \operatorname{argmax} \mathbf{E}^{\mathtt{f}} \hat{\mathbf{x}}_{\mathtt{cat}}^{\mathtt{f}}{}^T \tag{1.17}$$

where argmax is a function that retrieves the index with the highest value in a given vector and $h(w^f)$ corresponds to the index of the symbol w in the French vocabulary. It is important to note that w^f may not be the exact word "chat" but it could also be "chaton" or "félin" as previously stated.

[4]`https://github.com/facebookresearch/MUSE#ground-truth-bilingual-dictionaries`

This retrieval method is the same as retrieving the nearest neighbours of the source word embeddings based on the cosine similarity assuming our embeddings are of unit length. In our cross-lingual language model, we will inevitably use retrieval to cycle from one language to another. We also plan to use Cross-Domain Similarity Scaling (CSLS) introduced by Conneau et al. which prevents the hubness problem (Radovanović et al., 2010) and leads to better performances compared to conventional nearest neighbours approach as demonstrated by Dinu and Baroni. We depict the usual workflow of a word-to-word translation in the context of a sentence in Figure 1.1.

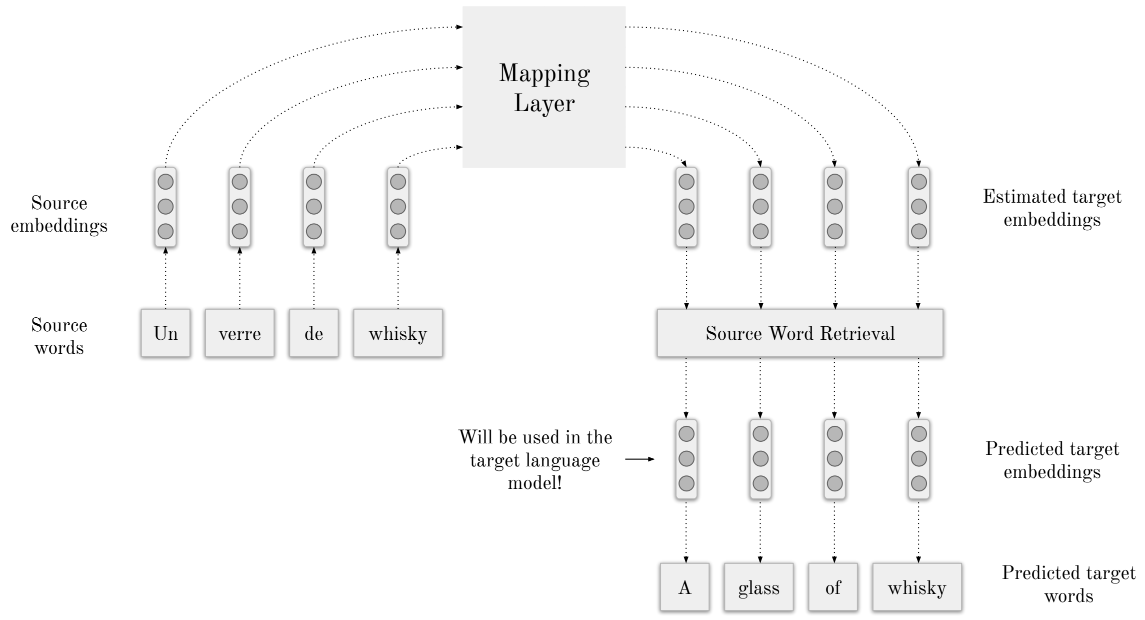


Figure 1.1 – An example of a word-to-word translation using a mapping based approach. Each word translation is independent from one another. We illustrate here the word-to-word translation using the scope of a sentence since it will be used in subsequent figures and help better understand the proposed approach of neural language model transfer.

This brings us to the evaluation of cross-lingual word embedding models. The intrinsic evaluation of cross-lingual word embeddings models is Bilingual Lexicon Induction (BLI); the goal is to determine the most appropriate translation from a word w_i^t for each query from w_i^s. To this end we will use the same datasets as Conneau et al. (2017) (lexicon[5] and embeddings[6]). The extrinsic evaluation lies in the downstream task; does the transferred language model helps to achieve better performance than less reliable pre-trained models or in the worst case, no model at all. This evaluation scheme is inherent in our proposed architecture in Section 3.3 since we will be using the transferred word representations in the source language model.

[5] https://github.com/facebookresearch/MUSE#ground-truth-bilingual-dictionaries
[6] https://fasttext.cc/docs/en/crawl-vectors.html

1.3 Neural Networks

1.3.1 Feed-Forward Neural Networks

A neural network is a mathematical function f that takes an input $\mathbf{x}$ (usually a vector or a matrix) and is composed of a set of learnable parameters Θ. In the simplest case, a neural network may take the form of a linear regression;

$$\hat{y} = f(x; \Theta) = x \cdot w + b \tag{1.18}$$

This simple network consists of a single neuron that takes as input x, has a set of parameters (w and a bias b), predicting a value $\hat{y}$. In this specific case, all values (x, y) and parameters (w and b) are scalars.

We can design a neural network with multiple neurons having multiple features as input using the vector/matrix notation as follows;

$$\hat{\mathbf{y}} = f(\mathbf{x}; \Theta) = \mathbf{x} \cdot \mathbf{W} + \mathbf{b} \tag{1.19}$$

where $\mathbf{x} \in \mathbb{R}^j$ is a vector of j features, $\mathbf{W} \in \mathbb{R}^{j \times l}$ represents l neurons consisting of j weights each and $\mathbf{b} \in \mathbb{R}^l$ is a vector of l biases. The final output $\hat{\mathbf{y}} \in \mathbb{R}^l$ thus consist of l values (in the case of a classification problem with l classes for example). $\mathbf{W}$ and $\mathbf{b}$ are the parameters of the neural network, which are represented as Θ in function f.

The parameters Θ are updated with the back-propagation algorithm (see Goodfellow et al. (2016)) by computing a loss function that we call $\mathcal{J}(\Theta)$ in this proposal. For a language model, the Cross-Entropy loss is most often used to compute the error of the neural network.

To add expressivity to a neural network, we introduce non-linear functions right after the application of a matrix $\mathbf{W}$ followed by the addition of a bias $\mathbf{b}$ on an input $\mathbf{x}$. We will then define a layer analogously as an f function as follows;

$$f_i(\mathbf{x}; \Theta_i) = \sigma(\mathbf{x} \cdot \mathbf{W} + \mathbf{b}) \tag{1.20}$$

We can then apply different functions f to an input $\mathbf{x}$;

$$\hat{\mathbf{y}} = f_2(f_1(\mathbf{x})) \tag{1.21}$$

We call this architecture “Feed-Forward” since the information flows linearly from the input $\mathbf{x}$ through the different layers f and outputs a vector $\mathbf{y}$. This architecture can be used as a language model by designing an input $\mathbf{x}$ that consists of k previous words and then predicts the next words $\hat{\mathbf{y}}$. This architecture will be detailed in section 2.3.1.

For the sake of simplicity, we refer the reader to the comprehensive Deep Learning book by Goodfellow et al. (2016) for the procedure in training a neural network (basically learning the set of parameters Θ), amongst other details.

1.3.2 Recurrent Neural Networks

Recurrent neural networks (RNN) is a specific architecture that handles sequential data. Natural language is a perfect case of sequential data; as we read, we sequentially process words to infer a specific sense of a given sentence.

They are designed to preserve a certain amount of history in memory while processing a given sequence (as humans normally do). They hold what is called a hidden state that is updated subsequently at each time step. One can think of a sentence (sequence of words) where each word is fed into the recurrent cell one at a time thus updating the internal state of the cell (and at the same time producing an output value) until the end of the sentence. We can thus see an RNN as an "Encoder" that encodes a sentence into a fix representation.

Vanilla recurrent neural networks depend on two inputs, $\mathbf{x}_t \in \mathbb{R}^j$ and $\mathbf{h}_{t-1} \in \mathbb{R}^l$, and are usually expressed in the following form;

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}; \Theta) = \mathtt{tanh}(\mathbf{W}_i \mathbf{x}_t + \mathbf{b}_i + \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{1.22}$$

where $\mathbf{W}_i \in \mathbb{R}^{j \times l}$ are the parameters that maps the input $\mathbf{x}_t$ to the hidden dimension, $\mathbf{W}_h \in \mathbb{R}^{l \times l}$ are the parameters that maps the previous hidden state $\mathbf{h}_{t-1}$ to a new intermediary hidden state and each of the input have their corresponding biases, $\mathbf{b}_i \in \mathbb{R}^l$ and $\mathbf{b}_h \in \mathbb{R}^l$. The `tanh` activation function is then applied to the resulting computation, which produces the hidden state at time t, $\mathbf{h}_t \in \mathbb{R}^l$.

RNN architectures are rarely used to model language in practice because they are hard to train (Pascanu et al. (2013)). A more sophisticated architecture that mitigates vanishing gradient is the Long Short Term Memory network (LSTM, Hochreiter and Schmidhuber (1997)), which we detail in Appendix A.

Using recurrent neural networks to model language is not new. Graves and Sutskever et al. demonstrated in the past that LSTMs are very effective to retain previous information in memory, thus modelling language. Mikolov's thesis also used such architectures to model language. Once an RNN is trained to model a language, it is able to **generate** textual data. It can thus be seen as a "Decoder", decoding words at each time step. Recent state-of-the-art RNN-based language models will be detailed in section 2.3.1.

Attention Mechanism

An attention mechanism is simply a function f that takes an arbitrary number inputs (from 1 to many) and outputs a score for each of these inputs. The mechanism introduced by Bahdanau et al. in the context of neural machine translation is composed of a single feed-forward layer f that takes as input a hidden state $\mathbf{h}_i^e$ that has been generated by the Encoder and the previous hidden state of the Decoder $\mathbf{h}_{t-1}^d$ which outputs a score e_i;

$$e_i = f(\mathbf{h}_i^e, \mathbf{h}_{t-1}^d). \tag{1.23}$$

The scores for each word in the source sentence conditioned on the Decoder's previous hidden state are concatenated and a softmax is applied to obtain the weight vector α. The conditioning context $\mathbf{c}_t$ is then the weighted sum of the hidden states of the Encoder $\mathbf{H}_{0:n}^e$;

$$\mathbf{c}_t = \sum_{i=0}^{n} \alpha_i \mathbf{h}_i^e \tag{1.24}$$

which is concatenated with the target language current word embedding $\mathbf{E}(w_t^d)$; $\mathbf{x}_t = \mathbf{E}(w_t^d) \oplus \mathbf{c}_t$. Note here that the conditioning context $\mathbf{c}_t$ is changing at each generation step of the Decoder. This allows the Decoder to pay attention to different parts of the source sentence during the generation of the sentence in the target language. Figure A.2 in Appendix A illustrate the behaviour of an attention mechanism in the context of machine translation using a RNN.

1.3.3 Transformer Neural Networks

Recently, a special kind of architecture has been proposed by Vaswani et al. for the machine translation application, the Transformer-based Neural Network. This architecture is a special kind of neural network largely inspired by the attention mechanism of Bahdanau et al. (2014) previously introduced. Similarly, it comprises an "Encoder" that encodes the source language's sentence and a "Decoder" that decodes the next word in the target language conditioned on the source sentence representation. We can in fact see the Decoder as a language model in the target language.

The "Transformer Encoder" begins by encoding the source sentence using a "self-attention" mechanism. The inputs $\mathbf{Q} \in \mathbb{R}^{n \times j}$, $\mathbf{K} \in \mathbb{R}^{n \times j}$, $\mathbf{V} \in \mathbb{R}^{n \times j}$ are called the *Query*, *Key*, *Value* respectively regardless it is the Encoder or Decoder. In the case of the Encoder, the *Query*, *Key* and *Value* are the same (self-attention). Thus, $\mathbf{Q} = \mathbf{K} = \mathbf{V}$ where the matrices represent the concatenation of n words' embeddings of dimension j that compose the source sentence [7]. The attention mechanism is expressed in the following way;

$$\mathbf{O} = \mathtt{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}})\mathbf{V} \tag{1.25}$$

[7] In practice, it is not literally the words' embeddings. They are in fact projected in different spaces, hence creating multiple *attention heads*

where $\mathbf{QK}^\top \in \mathbb{R}^{n \times n}$ represents n attention scores that each word in the sentence attributes to every other words (even itself) [8] . The reader can assume that $\sqrt{d_k}$ is a constant used for stability in the computation. The output $\mathbf{O}$ is further used in subsequent layers, creating "piles" of attention layers. We forward the reader to the paper of Vaswani et al. for details of the computations.

At the very end of the Transformer Encoder, an encoded representation of each word in the source sentence is generated which we call $\mathbf{Q}_d \in \mathbb{R}^{n \times j}$. In fact, this constitutes the conditioning element of the Transformer Decoder during the generation of the next word in the target sentence, often called the "intra-attention" mechanism between the Encoder and the Decoder. The Key $\mathbf{K} \in \mathbb{R}^{m \times j}$ and Value $\mathbf{V} \in \mathbb{R}^{m \times j}$ correspond to the m words' embeddings of the decoded word so far in the target sentence. The final output of the Decoder is the prediction of the next word that follows in the target sentence given the decoded words so far and the encoded source sentence.

In the case of the decoder, the attention matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ corresponds to the attention scores (i.e. importance of) for the n words in the source sentence attributed to the m decoded words so far in the target language. Figure 1.2 illustrates a one-layer version of the architecture for the sake of simplicity.

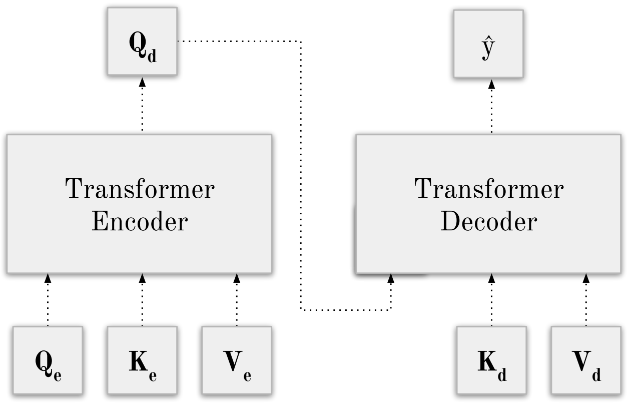


Figure 1.2 – Overview of the Transformer Architecture proposed by Vaswani et al. (2017).

It is now easy to understand that a Transformer-based language model is a Transformer Decoder that is conditioned only on the previous *decoded* words, thus performing only self-attention given the n previous words. Similar to the Transformer Encoder, with can then define the input of a Transformer-based language model as follows; $\mathbf{Q} \in \mathbb{R}^{n \times j}$, $\mathbf{K} \in \mathbb{R}^{n \times j}$, $\mathbf{V} \in \mathbb{R}^{n \times j}$. The only difference with the Transformer Encoder previously introduced is that a Transformer-based language model is not trained to encode a source sentence representation for later conditioning. It is only trained to encode the source sentence to predict the next word in that same sentence. State-of-the-art Transformer-based language models will be detailed in section 2.3.1.

[8]It is important to note that the values of $\mathbf{Q}$ are broadcasted which means each row of $\mathbf{Q}$ are multiplied with each row of $\mathbf{K}$.

Chapter 2

Literature Review

2.1 Language Models Granularity

Language models, as introduced in section 1.1, are able to predict the next symbol given a sequence of k previous symbols. We denote $\mathcal{V}_s$ the set of all possible symbols. These models may act on different granularity such as words, characters, n-grams of characters commonly called Byte-Pair Encoding (BPE) [1] or Word Pieces (Devlin et al., 2018) also known as subwords, yielding different vocabulary sizes. On a **word** level, a language model predicts with a certain probability the next word w_i given a sequence of k previous words;

$$P(w_i|w_{i-k}, \dots, w_{i-1}) \tag{2.1}$$

The number of words w in the vocabulary $\mathcal{V}_w$ can reach more than a 100,000 words (TransformerXL (Dai et al., 2019) has more than 200,000 symbols in its vocabulary) and thus the model may be prohibitive in memory requirements. A language model can also act on the character level, reducing considerably the vocabulary size. Such models then predict the next character c_i given a sequence of k previous characters;

$$P(c_i|c_{i-k}, \dots, c_{i-1}) \tag{2.2}$$

Another way to handle the vocabulary of a language model is to use BPE. BPE is an algorithm of data compression that replace the most common pair of consecutive bytes (characters in our case) with a special byte that does not occur within the corpus. It was first introduced by Sennrich et al. (2016) in the Neural Machine Translation task.

Recent architectures such as Radford (2018) also use the BPE algorithm to generate a vocabulary of reasonable size. We denote a byte-pair as $b_i = [c_{i-1}; c_i]$ where c_{i-1} and c_i are a common pair of consecutive characters and define a byte-pair language model as follows;

[1] Byte pair encoding [...] is a simple form of data compression in which the most common pair of consecutive bytes of data is replaced with a byte that does not occur within that data. (Wikipedia contributors, 2019)

$$P(b_i|b_{i-k},\dots,b_{i-1}) \tag{2.3}$$

Note that b_i could also be a single character. Usually, we cap the number of byte-pairs to around 40,000 symbols as Radford proposed.

2.1.1 Conditional Language Models

Fundamentally, language models are *conditional* language models; they model the probability to generate the next symbol conditioned on k previous symbols. But language models can be conditioned on many other things. It is possible to design a language model that reproduces a specific stylistic, model an author, or a specific domain of applications (Ficler and Goldberg, 2017).

One of the most famous applications of a conditioned language model is machine translation (MT). Bahdanau et al. condition the generation of the next word in the target language not only on the previous words generated from the model but also on all the words in the source sentence, which we call the context $\mathbf{c}$.

We can then define a conditional language model as follows;

$$P(s_i|s_{i-k},\dots,s_{i-1};\mathbf{c}_i) \tag{2.4}$$

Where $\mathbf{c}_i$ is the context at time i. We specifically write the context in bold type since it can contain multiple conditioning elements (author, style, sentiment, etc.).

Image captioning is an example of conditional language generation. It leverages the content of an image to generate a small description of it. Usually, an abstract representation vector of the image is obtained using a pre-trained convolutional neural network (He et al., 2016) which serves as the conditioning parameter $\mathbf{c}$. It is important to note that the context may change during the generation of a specific symbol, as in the model proposed by Bahdanau et al. for the task of machine translation.

Essentially, language models can be conditioned on any kind of contexts and they are usually used like so. We illustrate many examples of how a context $\mathbf{c}$ can be obtained and fed to a language model mathematically in Section 2.4.

2.1.2 N-Gram Language Models Limitations

One can be easily convinced that n-gram language models inhibit a fundamental problem: *they rely on a short range of previously seen symbols.*

Goldberg specifically demonstrated [2] that a character n-gram model can use up to 10 previous characters to predict the next one on the Shakespear corpora. More than this would result in

[2] `https://nbviewer.jupyter.org/gist/yoavg/d76121dfde2618422139`

a really small probability when comes the time to predict the next word given a long history of symbols.

In the case of a combination of words never seen in the training corpus, the probability is just unknown. Several smoothing methods are used to circumvent the unseen n-gram problem. One can use the Laplace smoothing (Manning et al., 2008, page 260) to remove probabilities of 0. Another solution is to do a linear interpolation between different n values of n-grams (unigram, bigram, etc.) by taking the weighted mean. These methods work well in practice but are limited in flexibility. We cannot think of adding a conditioning context $\mathbf{c}$ in n-gram language models since it will inevitably drop all probabilities of seing the next symbol to zero.

While the results of n-grams language models are syntactically interesting, there is a major lack of coherence throughout the generation of a long piece of text. Neural networks have proven to better handle a longer range of previous k symbols which helps preserve coherence and at the same time being able to easily condition a language model on a given context $\mathbf{c}$ without the need for smoothing methods. We specifically detail language models using different neural network architectures in Section 2.3.

2.2 Evaluation of Language Models

Language models are able to estimate to probability of a given sequence of symbols. In practice, we don't use the probabilities but the perplexity which is a metric that is commonly used to evaluate *intrinsically* a language model. The perplexity is the inverse probability computed over a complete test set, normalized by the number of symbols. Given a sequence T composed of n symbols, the perplexity of such a set is given by this formula;

$$\texttt{Perplexity}(T) = P(s_1, s_2, \ldots, s_n)^{\frac{1}{n}} \tag{2.5}$$

where the lower the perplexity the better the language model. It can be seen as "how surprising" a textual utterance is. While this metric gives a good indicator of the performance of a language model and is widely used in a generic evaluation setting (Melis et al., 2018; Merity et al., 2018), it may not represent it faithfully. At this moment, it may be more meaningful to evaluate a language model within a specific natural language processing task such as Machine Translation (Bahdanau et al., 2014), and Question Answering (Yin et al., 2016) by measuring the effective gain of performance. We propose in Section 3.2 to evaluate a multimodal neural language model in a conversational context.

Several of these downstream tasks (Machine Translation, Dialogue Systems) rely on two other metrics that can be used to evaluate a language model. The first one is the BLEU score (Papineni et al., 2002) which evaluates the quality of an automatically translated text compared to a human's translation. This metric is based on the overlapping unigrams and bigrams between two sequences, thus capturing the precision of a translation. Another metric (ROUGE) has been proposed (Lin, 2004) and uses several combinations of n-grams to compute an overall score of a given generated sentence and its corresponding ground truth. This metric rather captures the recall of a particular generation.

However, these metrics are not always well suited for the evaluation of automatically generated language. Liu et al. showed that BLEU and ROUGE scores are loosely correlated with human judgment when comes the time to evaluate dialogue response generation. Also, these metrics can only be used with conditional language modelling and when the number of plausible target sentences is limited.

Another form of evaluation is based on word embeddings. Given a ground truth sentence $\mathbf{S} \in \mathbb{R}^{n \times d}$ composed of n word vectors $\mathbf{w}$ where $\mathbf{w} \in \mathbb{R}^d$ and a predicted sentence $\hat{\mathbf{S}}$ composed of the same number of word vectors, one can compute a similarity between these two sentences using various metrics. One way to compute the similarity between two sentences is to use the average of the embeddings for each sentence and then compute the cosine similarity between the two averaged representations $\mathbf{s}$ and $\hat{\mathbf{s}}$ in the following fashion;

$$\begin{aligned} \mathbf{s} &= \frac{1}{n}\sum_{i \in n} \mathbf{w}_i \\ \hat{\mathbf{s}} &= \frac{1}{n}\sum_{i \in n} \hat{\mathbf{w}}_i \\ sim &\leftarrow \text{cosine\_similarity}(\mathbf{s}, \hat{\mathbf{s}}). \end{aligned} \tag{2.6}$$

The exact same sentences will produce a similarity of one while two completely different sentences produce a lower similarity score around 0.2-0.4.

This leads to one last form of evaluation that is more reliable but a lot more costly than the previous ones; human evaluation. This metric of evaluation has serious disadvantage such that it is not automatic thus taking a lot of time to perform. It is also hard to give a rating to a generated sentence of paragraph. However, a human can easily spot inconsistent sequences of words of incoherent paragraphs generated by a language model.

Given all these possible ways to evaluate language models, every method has its drawbacks and there is no magic way to properly assess the quality of automatically generated text [3]. Either way, this proposal is going to rely mainly on perplexity and human evaluation as an evaluation metric across research avenues.

2.3 Neural Language Models

Neural language models have been thoroughly studied in the past years. Different architectures have been proposed as well as their limitations exposed (Jozefowicz et al., 2016) and this section is dedicated to introduce the most notable ones.

[3]This could be, in fact, a whole research proposal.

We refer the reader to Appendix B for in-depth training details that need to be taken care of in order to obtain state-of-the-art language models. These details are often omitted in publications.

2.3.1 Models Variety

We begin this subsection by presenting a strong baseline using a feed-forward neural architecture that has been introduced in section 1.3.1. RNN, presented in section 1.3.2, have been for a long time the state-of-the-art in language modelling and are thus presented next. We then conclude this subsection with the new state-of-the-art architectures used to model languages, the Transformer-based (Attentional) language models that were introduced in section 1.3.3.

Feedforward Language Models

As stated in section 1.3.1, feed-forward neural networks are a mathematical function f that takes as input a vector $\mathbf{x} \in \mathbb{R}^d$ parameterized by $\mathbf{W} \in \mathbb{R}^{d \times |\mathcal{V}|}$ and $\mathbf{b} \in \mathbb{R}^{|\mathcal{V}|}$, a set of learnable weights. In the context of language modelling, we properly define our input $\mathbf{x}$ based on a *fixed size* of k previous words need to predict the nth word;

$$\mathbf{x} = \mathbf{w}_{n-k} \otimes \ldots \otimes \mathbf{w}_{n-1} \tag{2.7}$$

where $\otimes$ is the concatenation operator and $\mathbf{w}$ are word embeddings obtained from a lookup table $\mathbf{E}$. If each word embeddings are composed of 100 components, the resulting dimension of the input $\mathbf{x}$ will be of size $k \times 100$. In its simplest form, a one-layer feed-forward neural language model would be expressed in the following way;

$$\begin{aligned}
\mathbf{x} &= \mathbf{w}_{n-k} \otimes \ldots \otimes \mathbf{w}_{n-1} \\
\mathbf{h} &= \sigma(\mathbf{x}\mathbf{W} + \mathbf{b}) \\
\mathbf{y} &= \text{softmax}(\mathbf{h}) \\
\hat{y} &= \text{argmax}(\mathbf{y})
\end{aligned} \tag{2.8}$$

where $\mathbf{w} \in \mathbb{R}^d$ $\mathbf{W} \in \mathbb{R}^{k*d \times |\mathcal{V}|}$, $\mathbf{b} \in \mathbb{R}^{|\mathcal{V}|}$, $\mathbf{h} \in \mathbb{R}^{|\mathcal{V}|}$ and σ denotes a non-linear function.

Similar to the Equation 1.19, we obtain a feed-forward language model that generates a vector $\hat{\mathbf{y}} \in \mathbb{R}^{|\mathcal{V}|}$ which corresponds to the score of each word in the vocabulary. We call these scores the *logits*. This model (as well as most of the next architectures presented in this proposal) is trained with the Cross-Entropy loss. Even though these models seem rather simple, they are a pretty strong baseline in the task of language modelling as demonstrated in the work of Bengio et al. (2003).

Recurrent Language Models

Similar to Goldberg (2017), we hereby lay a specific notation for RNN-based LM. We use $\mathbf{x}_{i:j}$ to denote a sequence of word embeddings. An RNN-based LM is thus a function that takes as

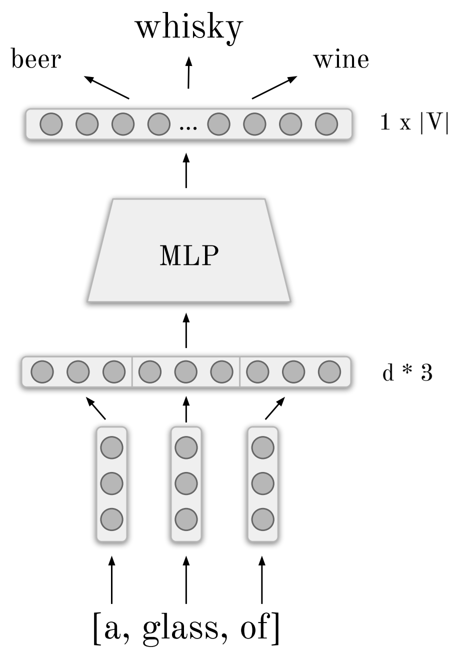


Figure 2.1 – A Feedforward Language Model. Taking as input the concatenation of each word's embedding $\mathbf{w} \in \mathbb{R}^d$ and outputs a score for each word in the vocabulary $\mathcal{V} \in \mathbb{R}^{|\mathcal{V}|}$. In this case, the window of k previous words to predict the next one is of size 3.

input an arbitrary length ordered sequence $\mathbf{x}_{1:n}$ of n d_{in}-dimensional embeddings and returns a sequence of d_{out}-dimensional vectors $\mathbf{h}_{1:n}$, parameterized by Θ;

$$\mathbf{h}_{1:n} = f(\mathbf{x}_{1:n}; \Theta) \tag{2.9}$$

One of the very first RNN-based LM was proposed by Sutskever et al. (2011), where they were using a vanilla RNN for text generation. The current state-of-the-art RNN-based LM is held by the `AWD-LSTM` [4] architecture (based on the LSTM cell introduced in section 1.3.2) proposed by Merity et al. (2017a) This architecture introduced several regularization techniques specifically for language models.

One thing to consider in the RNN-based language models is that the hidden state $\mathbf{h} \in \mathbb{R}^d$ that is generated at each time step is never the same size as the vocabulary (usually much lower). This creates the need to add a specific *mapping* layer $\mathbf{M} \in \mathbb{R}^{d \times |\mathcal{V}|}$ that takes as input the hidden state $\mathbf{h}_n$ and outputs a vector $\mathbf{y} \in \mathbb{R}^{|\mathcal{V}|}$ which in fact adds a lot of parameters to the architecture;

$$\mathbf{y}_t = \mathbf{h}_t \mathbf{M} \tag{2.10}$$

[4] Several improvements have been proposed recently (Gong et al., 2018; Takase et al., 2018; Wang et al., 2019; Melis et al., 2020).

Press and Wolf proposed to tie the parameters of the embedding layer $\mathbf{E}$ with the mapping layer $\mathbf{M}$ which enforces $\mathbf{M} = \mathbf{E}^\top$. It also forces $\mathbf{h} \in \mathbb{R}^{d_{embed}}$ where d_{embed} is the dimension of the embeddings. Indeed, it offers the opportunity to share parameters across layers thus reducing the number of parameters significantly.

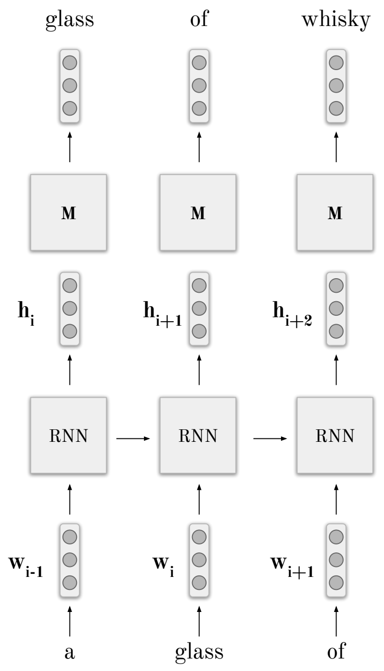


Figure 2.2 – A Recurrent Language Model. Taking as input one word embedding $\mathbf{w} \in \mathbb{R}^d$ at a time and outputs a hidden state $\mathbf{h}_{i+1}$ that is fed into the mapping layer $\mathbf{M}$ which gives a score for each word in the vocabulary $\mathcal{V}$ which is used to predicting the next word. The window size a Recurrent Language Model is unbounded, contrary to Feedforward and Transformer language models.

Training an RNN-based language model can be long since the parallelization of RNN is difficult. The training algorithm, called Back Propagation Through Time (BPTT), is used to update the weights of the recurrent cell. BPTT works by unrolling the recurrent neural network, as shown in Figure 2.2. For each input, there is an output prediction. At the end of the sequence, a loss is computed over every time steps and then the weights are updated. Suppose we have this very long sentence;

```
Nicolas walks towards the bar after meeting is good old friend
Jean-Simon, whom he hasn't seen in a long time, to order some salted
peanuts and a glass of whisky.
```

Only triggering the back-propagation algorithm at the end of long sequences like this one would prevent the recurrent neural network to update its weights often, thus learning slowly the natural aspect of a language.

However, a judicious technique called Truncated Back Propagation Through Time (Truncated

BPTT) accelerate the process of training such architectures. Instead of only triggering the back-propagation algorithm at the last time step (t_n), we split the inputs into k-length sequences of symbols where $k < n$ allowing the update of the parameters of the network much more often. This leads a faster training of such architectures. Usually ranging from 10 to 100 symbols. We illustrate the difference between BPTT and truncated BPTT in Figure 2.3 and Figure 2.4 respectively.

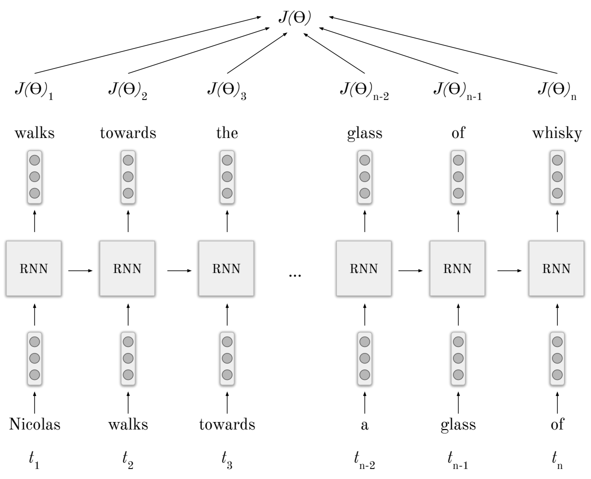


Figure 2.3 – Standard Back Propagation Through Time, the back-propagation algorithm is triggered at the time step t_n.

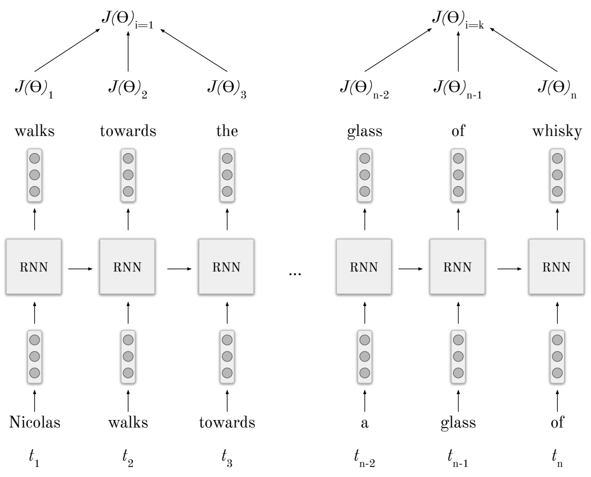


Figure 2.4 – Truncated Back Propagation Through Time (b) triggers the back-propagation algorithm more often. In the example, it is triggered at every 3 symbols processed by the recurrent cell, $t_3, t_6, \ldots, t_n$.

We expose the detailed equations of an RNN-based language model in order to fully grasp the understanding of the process at each time step;

$$
\begin{aligned}
\mathbf{x}_t &= \mathbf{E}(w_t) \\
\mathbf{h}_t &= \texttt{RNN}(\mathbf{x}_t; \mathbf{h}_{t-1}) \\
\mathbf{y}_t &= \mathbf{h}_t \mathbf{M}
\end{aligned}
\tag{2.11}
$$

where $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the embedding lookup matrix, $\mathbf{x}_t, \mathbf{h}_t, \mathbf{h}_{t-1} \in \mathbb{R}^d$ and $\mathbf{M}^{d \times |\mathcal{V}|}$ which corresponds to (Press and Wolf, 2017) as $\mathbf{E}^\top$.

Transformer-based Language Models

There has been a huge body of work with Transformer-based language models in the last two years. These models are mainly composed of feed-forward layers [5].

The first introduced Transformer-based LM is proposed by Liu et al. where they cherry-picked only the decoder from Vaswani et al. (2017). It is important to note that unlike the RNN-based LM, where at each time step only the i-th symbol is processed in order to predict the $i+1$-th symbol, Transformer-based LM models needs to reprocess the whole sequence of $1:i$ symbols at each time, due to the absence of recursion.

Also, since there is no recursion in the Transformer model, a positional encoding is added to the word's representation. We explicit the inputs and outputs in Equation 2.12. For the sake of simplicity, we refer the reader to the paper of Vaswani et al. for in detail explanations of the Transformer architecture.

$$
\begin{aligned}
\mathbf{Q}_{1:i}, \mathbf{K}_{1:i}, \mathbf{V}_{1:i} &= \mathbf{E}(w_{1:i}) + \text{PositionalEncoding}(w_{1:i}) \\
\mathbf{H}_{1:i} &= \text{Transformer}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \\
\mathbf{Y}_{1:i} &= \mathbf{H}_{1:i} \mathbf{M}
\end{aligned}
\tag{2.12}
$$

An improvement of the last model (Vaswani et al., 2017) has been proposed by OpenAI (Radford, 2018) where they doubled the amount of attention layers (from six to twelve) and applied their model on a series of NLP downstream tasks as well as evaluating on several language modelling benchmark datasets such as (Wang et al., 2018).

The two previous models have been applied to the BPE level and a similar model (Al-Rfou' et al., 2018) has been applied on the character-level yielding near-state-of-the-art perplexity results on enwik8 (Mahoney 2009) which is the first 10^8 bytes (compressed) of the English Wikipedia dump on Mar. 3, 2006 [6].

A *masked* language model (MLM) has been proposed by Devlin et al. (2018) which its goal is not to model language specifically, whereas *causal* language models (CLM) previously introduced do, but it has been shown by Wang and Cho (2019) that it is able to generate text as a standard language model should.

[5] Some implementations use 1D convolutions which is similar to a feed-forward layer.

[6] https://cs.fit.edu/~mmahoney/compression/text.html

A more sophisticated model was introduced by Dai et al. which handles longer sequences of symbols by introducing a recursive mechanism. They achieved state-of-the-art results on WikiText-103 (Merity et al., 2017b), enwiki8, text8, One Billion Word (Chelba et al., 2013) and Penn Treebank (Marcus et al., 1994).

Finally, OpenAI demonstrated that training a large-scale unsupervised language model (Radford et al., 2019) on a huge body of text can generate coherent paragraphs of text and achieve state-of-the-art performance on many language modelling benchmarks. This model can also perform rudimentary reading comprehension, machine translation, question answering, and summarization, all without task-specific training. Their model is largely inspired by the first version of Radford (2018), using the same Transformer-Decoder architecture as Liu et al.. The key idea is that they use a specific input flag (called the delimiter) depending on the target task.

2.4 Neural Language Generation

Language models can be used to evaluate the likelihood of a piece of text but they can also be used to **generate** text, taking the form of generator or **Decoders**. In this section, we will present two forms of generation, free form generation as well as conditioned generation. Free form generation takes place when we want to generate open domain textual data. Usually, we will use **conditional** language models, trying to achieve a particular task. Conditional language models, as introduced in Section 2.1.1, play an important role in many downstream tasks. We first present traditional free-form generation and then present different ways to condition a language model as well as different conditioning sources.

2.4.1 Free Form Generation

One of the main utility of a language model is to generate text. As stated in Section 2.1.1, language models are implicitly conditioned on the previous symbols. In a free-form generation (which we call here *unconditioned generation*), the model uses only the previous k symbols to generate the next one in contrast of using an additional conditioning input $\mathbf{c}$ which is discussed next.

To begin, we start with a special `START` symbol that initiates the generation. If we want to have a deterministic generation (which won't have any variety as it always returns the same generated sentence), we take the next most probable symbol given the `START` symbol. On the next step, we condition the model using the new symbol that we picked concatenated with the `START` symbol. We can iterate through this loop until we hit the `END` symbol.

The fact that there is potentially an infinity of ways to write things down makes the evaluation of a generated textual utterance a hard challenge to automate. Indeed, a human could carefully look at all the generations done by a language model but in practice it is not viable if we want to rapidly iterate in the process of developing our language model.

Instead, we evaluate the trained model on a corpus that has been written by humans. As stated in Section 2.2, we use the perplexity (see Equation 2.5) to assess the likelihood of a corpus according to the trained model.

The best architecture at modelling language so far is GPT-2 (Radford et al., 2019), the second version of the Generative Pre-Training method proposed by Radford. It achieves the best perplexity on the Penn Treebank (Marcus et al., 1994), Enwiki8, Text8 and WikiText-2 (Merity et al., 2017b) datasets.

In the next section, we will see other applications of language models and how we can evaluate them differently with downstream tasks.

2.4.2 Conditioned Generation

Conditioned generation generally implies a downstream task; we condition our language model not only on the previous symbols but also on additional information that serves for a specific application. The performance of these language models is evaluated extrinsically by measuring the performance of the overall architecture on the downstream task.

Conditioning is usually done by concatenating the input embedding of $\mathbf{w}_t$ with a contextual vector $\mathbf{c}_t$;

$$\mathbf{x}_t = \mathbf{E}(w_t) \oplus \mathbf{c}_t \tag{2.13}$$

where $\mathbf{x}_t$ now constitutes the input to the decoder.

We will then present a non-exhaustive list of applications relevant to this research proposal where language models play an important role as well as how the conditioning information is integrated (formerly how $\mathbf{c}$ is built) within the language model.

Machine Translation

Machine Translation (MT) is the task of translating a source sentence $\mathbf{s}_s$ of length n to a target sentence (in another language) $\mathbf{s}_t$ of length m. A standard neural approach uses a RNN Encoder-Decoder architecture where the Encoder is an RNN (usually an LSTM) that encodes $\mathbf{s}_s$ into a fixed length vector called the context $\mathbf{c}_s$. The Decoder (which is another RNN) starts generating the target sentence using $\mathbf{c}_s$ as the initial hidden representation ($\mathbf{h}_0^d$) and the `START` token until the `END` token is generated. This conditioning scheme works on short sequence but fall short to handle longer ones.

As introduced in Section 1.3.2, Bahdanau et al. introduced a simple yet effective **attention mechanism** which makes possible the processing of longer sequences.

More recently, Vaswani et al. introduced the Transformer architecture (see Section 1.3.3) in the MT task, completely removing the recurrence scheme of the RNN-based language models. Still, this model uses an Encoder-Decoder architecture where the Encoder encodes the source sentence and the Decoder generates the target one. The conditioning is introduced using "intra-attention", where the keys $\mathbf{K}$ and values $\mathbf{V}$ come from the Encoder's source sentence representation and the query $\mathbf{Q}$ changes during the generation, coming from the Decoder at each time step.

We depict the process of decoding a target sentence given a source sentence in Figure **??**. For the sake of simplicity, we only illustrate the matrix computations of the different inputs $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ in a one-layer Transformer architecture.

The inputs have three components. In this example, the query $\mathbf{Q}_d$ coming from the decoder consists of the first three words in the target sentence, `<s>`, `Je` and `mange`. $\mathbf{K}_e$ and $\mathbf{V}_e$ represent the source sentence, `I eat an apple`, which we want to translate. The first computation results in the attention scores matrix $\mathbf{A} = \mathbf{Q}_d\mathbf{K}_e^\top$. This allows every word in the target sentence pay attention to every other word in the source sentence to generate the next target word. The result is then multiplied with $\mathbf{V}_e$, yielding the query for the next layer $\mathbf{Q}'_d$ or the prediction of the next word if it is the last layer in the architecture.

In both cases (Bahdanau et al., 2014; Vaswani et al., 2017), their conditional language models have been evaluated on the BLEU score using the newstest2015 from the Europarl corpus [7].

Generating Text from Structured Data

From now on we have seen language generation without much restriction on the content, only conditioning on the previous symbols or a given theme. Generating text from structured data (also known as *data-to-text*) is an application where a generator has to output pieces of text where specific information **needs** to be said. This information could be represented as a checklist or a table, for example.

Kiddon et al. proposed the *neural checklist model*, an RNN that models global coherence by storing and updating an agenda of text strings which should be mentioned somewhere in the output. The model generates output by dynamically adjusting the interpolation among a language model and a pair of attention models that encourage references to agenda items. Their model is conditioned on a specific goal g, an agenda of items that needs to be said $\mathbf{E} = \{\mathbf{e}_1, \ldots, \mathbf{e}_{|\mathbf{E}|}\}$, a checklist of soft records of what items have been used $\mathbf{a}_{t-1}$, and the previous k symbols encoded in the previous hidden state h_{t-1}. The input to the neural language model is the sum of the projected input (mainly for dimensional purposes). They evaluated their model on a dataset of recipes where a model needs to write the steps to prepare a dish given the ingredients. They used the BLEU score as well as human evaluation to assess the quality of the generated recipes.

Moryossef et al. proposed a model that structures the information that has to be written (planning) before generating fluent language describing that information (realization). The planning model generates a plan that is used within a vanilla neural machine translation (NMT) system (similar to Bahdanau et al. (2014)). The NMT system takes as input the linearized plan and outputs a target sentence. Again, they evaluated their model with the BLEU score and human judgment.

[7]`https://www.statmt.org/europarl/`

Lebret et al. introduced a neural language model conditioned on data tables from Wikipedia to generate biographical texts. They used a feed-forward neural language model (see Section 2.3.1) that takes as input word embeddings concatenated with different attributes embeddings referring to table elements similar to the notation introduce with the work of Ficler and Goldberg (2017). When a specific word is not associated to a table element, they simply used an empty embedding. Bao et al. performed the same task and basically used the same model as Bahdanau et al. to pay attention to the different table attributes during generation. In both cases, they used the BLEU score to evaluate their architecture.

These data-to-text method will be further discussed in the second proposed research avenue, Section 3.2.

Template Based Neural Language Generation

Template based language generation is essentially factored in two steps; the first step is to find what to say, and then how to say it. It is more considered a paradigm of generation than a task per se. The kind of architecture used often implies an Encoder-Decoder (Sutskever et al., 2014).

Wiseman et al. used this technique on a data-to-text corpus, trying to generate sentences given structured data from Wikipedia. They proposed a method that learns templates from a training corpus. The RNN decoder is then conditioned on these learned templates to sort of "fill in the blanks" during generation. The decoder accepts as input the current symbol $\mathbf{x}_t$ and a current state $\mathbf{z}_t$ that contains soft information about the templates learned from the training set;

$$\mathbf{y}_{t+1} = \texttt{RNN}(\mathbf{x}_t, \mathbf{z}_t; \mathbf{h}_{t-1}). \tag{2.14}$$

The interesting feature of this approach is that the latent vector $\mathbf{z}_t$ may generate several symbols before changing, thus changing the template during generation. They showed that the generation is much more interpretable than traditional neural language generators and offers more controllability. They evaluated their model on BLEU for each of their experiment.

Peng et al. also applied this method on data-to-text and abstractive summarization. The major difference with template-based generation is that instead of filling in the blanks, they use the full example extracted from the training to generate a new textual utterance that is similar to it using the copy mechanism Gu et al. (2016), called *exemplar*. The trick behind their approach is a reparameterization step; during the generation of a textual utterance, they change the parameters of the decoder according to some examples extracted from the training corpus. To illustrate their model, they use a simpler notation, the Elman network (Elman, 1990);

$$\mathbf{h}_t = \texttt{tanh}(\mathbf{Q}\mathbf{x}_t + \mathbf{P}_t\mathbf{h}_{t-1}) \tag{2.15}$$

where **Q** and **P** are dynamically computed given exemplars extracted from the corpus, thus conditioning the generation. They evaluated the performance of their approach on ROUGE for each dataset used.

Image captioning

Image captioning is the task of describing an image with textual information which implies a language model. It thus combines two major artificial intelligence; computer vision and natural language processing. A popular dataset used to tackle this problem is Microsoft COCO (Lin et al., 2014), containing more than 300,000 images with 5 captions each.

You et al. proposed a model which uses a convolutional neural network that proposes concepts extracted from the image that form the conditioning context to the recurrent neural language model. They employ a semantic attention model to combine the visual features with visual concepts in the decoder that generates the image caption. The encoded visual features are then concatenated with the language model's input word. We illustrate the model in Figure 2.5 where the CNN architecture could be any pre-trained model such as He et al. (2015).

In the same line of research, Liu et al. used a region proposal convolutional neural network such as Girshick et al. (2013) that proposes textual concepts extracted from the input image. They used a self-attention mechanism (Vaswani et al., 2017) along with a recurrent neural language model to generate the caption.

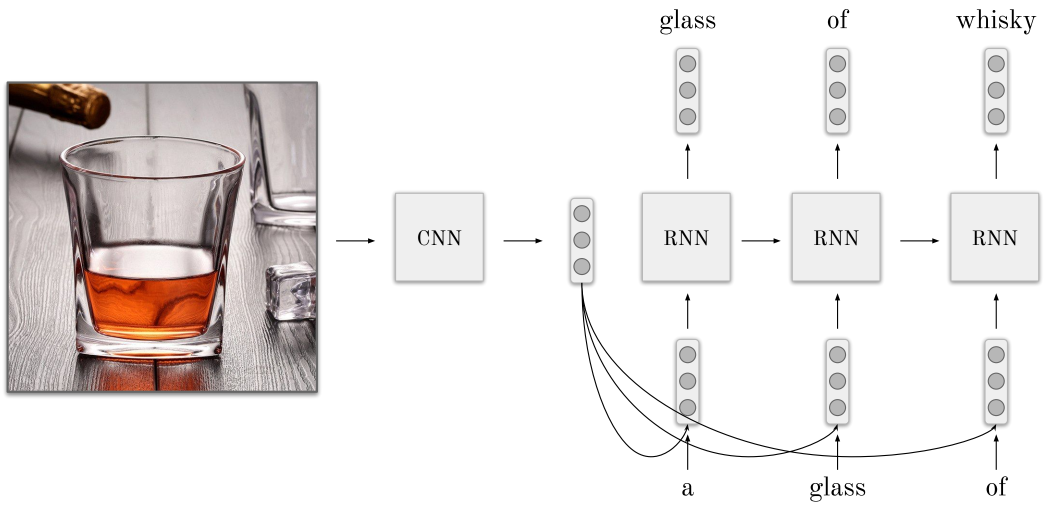


Figure 2.5 – An image captioning standard architecture. The extracted semantic representation of the image is concatenated at every time steps during the generation of the caption.

Dialogue Systems

Dialogue systems is the task of teaching a machine to converse naturally with humans. These systems can roughly by categorized into 2 groups; non-task-oriented systems also known as chat bots and task-oriented systems. The former one focus only on conversing with humans, providing reasonable answers to open domain questions. Hence, the language model learns to generate responses conditioned on the history of the dialogue. The latter one highly depends on a knowledge base related to the task we are trying to accomplish (e.g. booking a restaurant). This language model is thus conditioned not only on the history of the conversation but also on structured data, similar to the data-to-text setting.

Wen et al. treat task-oriented dialogue as a sequence to sequence mapping problem similar to Sutskever et al. (2014) augmented with the dialogue history and the current database search outcome (either looking for food, phone number, address, etc.). They use an LSTM as the generator that takes as input the last output token w_t and an encoded representation of the conditioning context $\mathbf{c}_t$ called the "action vector". They evaluated the performance of their approach with the BLEU score as well as human evaluation.

Li et al. and Shao et al. proposed two models working on non-task-oriented dialogue systems. Li et al. (2017) simply encoded the history of the conversation as a conditioning factor while Shao et al. (2017) use an attention mechanism to generate longer and more diverse responses.

The Utility of Language Models

As we can see, language models are inherently present in several natural language understanding tasks from the free-form generation to dialogue systems. Conditioned language models offer a better way to evaluate their performance since the quality of the generation can be assessed according to the target task. While several advancements have been done regarding these models, there is still plenty of work to do and we will tackle different research avenues in order to improve the generalization of pre-trained language models the next chapter.

Chapter 3

Research Avenues

In this chapter, we propose 3 main research avenues. The first one provides an algorithm to transfer a pre-trained language model towards another language that potentially has fewer resources to be trained on. The purpose of the transferred language model will be to generate text in the target language by leveraging the syntactic and semantic capabilities learned by the source neural language model. Since they are the actual state of the art in the textual generation, we plan to use Transformer-based language models (Vaswani et al., 2017).

The second research avenue focuses on conditioning such Transformer-based language models. So far, these models have only been used in a textual conditioning setting (up to a few hundred symbols in most cases) in every natural language understanding tasks. We propose to extend the conditioning of these models so that they can handle a greater diversity of conditioning elements such as whole documents, images, and even structured data. The evaluation of this new conditional neural language model architecture will be done using a domain-specific dataset that comprises different conditioning elements in the context of question answering.

The third research avenue focuses on grounding multilingual neural language models using structured data. In fact, we want to study the learning dynamics of mBERT, the multilingual version of BERT (Devlin et al., 2018) using a new cross-lingual analogy dataset. We propose a four-way Siamese model trained on an analogy task which we plan to evaluate on a multilingual benchmark (Hu et al., 2020).

In a matter of completeness of this proposal, we also include technical details in Appendix D which introduce every tool and methodology we wish to use in order to conduct our experiments.

3.1 Generalizing Neural Language Generation to Other Languages

The web comprises an outstanding volume of textual data and we just see the rise of larger and larger models such as ELMo (Peters et al., 2018), BERT (Devlin et al., 2018), TransformerXL (Dai et al., 2019), GPT-2 (Radford et al., 2019), XLNet (Yang et al., 2019), and the recent training scheme proposed by Facebook, RoBERTa (Liu et al., 2019b) just to name a few.

These pre-trained language models can be transferred to many natural language understanding tasks, outperforming the previous state-of-the-art models. This is a turning point in the natural language processing community, similar to the “ImageNet moment” we saw in the computer vision community in the early 2010s. Peters et al. specifically demonstrated that these pre-trained language models are inherently feature extractors with which a simpler classifier can be used to achieve a specific task (e.g. text classification or sequence labelling).

While these models achieve outstanding performances on a specific language[1], transferring them to another language is still an active field of research. We thus present in the following section the work that has been done to date related to the cross-lingual transfer of pre-trained language models. We further propose to explore the cross-lingual transfer of pre-trained language modelling specifically for **causal language modelling** (CLM) without or with minimal supervision, where the focus is to improve language modelling, hence language generation, on low-resource languages.

3.1.1 Related Work

Transferring a neural language model towards another language's specific task has only been merely explored recently. A method proposed by Lample and Conneau (largely inspired by Klementiev et al. (2012)) jointly learns cross-lingual representations in individual languages. Concretely, they have a language model that is trained simultaneously on two different languages which we call the source languages. This model is then transferred towards a third target language for fine-tuning to accomplish different natural language understanding tasks such as text classification or sequence tagging. Their model requires a large amount of data in all languages to achieve interesting results in the downstream tasks.

One of the most promising architectures to date that achieved interesting results on the cross-lingual transfer of neural language models is without a doubt Multilingual BERT (Devlin et al., 2018). The analysis done by Pires et al. shows that a Multilingual BERT model (mBERT) pre-trained on 104 different languages with standard BERT's training objectives can generalize to several other languages in a somewhat “Zero-Shot” setting (without further training of the model on the downstream task's data). They evaluated the transferred architecture on two different sequence tagging tasks; Named Entity Recognition (NER) on two datasets (Sang, 2002; Sang and Meulder, 2003, CoNLL-2002 and CoNLL-2003 containing Dutch, Spanish, English, and German) and Part of Speech Tagging (POS) using 41 languages of the Universal Dependencies dataset (Nivre et al., 2016).

However, Virtanen et al. and de Vries et al. showed that for a specific language like Finnish or Dutch, mBERT is not enough, necessitating a monolingual model for the particular classification tasks. While mBERT works well on several classification tasks and languages, Rönnqvist et al. demonstrated that under really simplistic configurations mBERT falls short to generated meaningful short pieces of textual utterances, which strengthen our point to better explore

[1]Mainly English but we now see pre-trained models in other languages too (Martin et al., 2019)

cross-lingual neural language generation.

The architecture from which we inspire ourselves and motivates this research proposal is the one proposed by Artetxe et al.. Based on the findings of Pires et al. (2019), they propose to transfer a monolingual neural language model (i.e. English BERT) towards another completely different language. Their approach does not rely on a shared vocabulary nor joint training. They showed that deep monolingual models learn some abstractions that generalize across languages. They had competitive results with mBERT on standard cross-lingual classification benchmarks. However, their model hasn't been applied to neural language generation, mainly due to the nature of the transferred model (BERT used the MLM objective) and still requires a substantial amount of training data.

To date, cross-lingual transfer of neural language model has mainly been applied to classification tasks, using BERT as the base architecture where its training objective (MLM) is not well suited for text generation. We thus propose to use pre-trained models that are well suited for natural language generation like GPT-2 (Radford et al., 2019) or TransformerXL (Dai et al., 2019) that uses a proper training objective (CLM), which we detail in the next section.

3.1.2 Research Proposal

Cross-lingual transfer for neural language as yet to be explored. Previous methods (Rönnqvist et al., 2019) only investigated textual generation using a pre-trained multilingual model that is trained with the Masked Language Modelling (MLM) objective (Devlin et al., 2018) with no success. We come to the conclusion that there is inevitably a need to investigate the transferability of neural language models architectures that have been specifically designed to generate text (Radford et al., 2019; Dai et al., 2019) using the Causal Language Modelling objective function (CLM).

In a matter of environmental efficiency, computational and textual resources, we propose a model transfer method designed to require minimal to zero supervision whereas we only use independent pre-trained modules to satisfy the task of cross-lingual neural generation of text. The proposed architecture will be presented in the next sections but we will introduce beforehand how we are going to use a word-to-word translation model as previously seen in Section 1.2.1 which constitutes an important building block of our proposed methods.

Word-to-Word Translation

We introduced in Section 1.2.1 how one can perform word-to-word translation using two sets of pre-trained word embeddings. We hereby assume that in our architecture we have a set of pre-trained word embeddings $\mathbf{E}^t \in \mathbb{R}^{|\mathcal{V}_t| \times n}$ in the target language such as the ones provided by Facebook [2] and that we have access to the source neural language model's embeddings $\mathbf{E}^s \in \mathbb{R}^{|\mathcal{V}_s| \times m}$. We also need a word-to-word dictionary (also provided by Facebook[3]). Given these resources in hand, we can learn a mapping (called the translation layer) from the source

[2] `https://fasttext.cc/docs/en/crawl-vectors.html`
[3] `https://github.com/facebookresearch/MUSE#ground-truth-bilingual-dictionaries`

to the target language $\mathbf{T}_{s\to t} \in \mathbb{R}^{m\times n}$ and, vice versa, a mapping from the target language towards the source language $\mathbf{T}_{t\to s} \in \mathbb{R}^{n\times m}$.

Cross-Lingual Language Model Transfer

Given a standard pre-trained monolingual neural language model, `LM`, we hereby lay the algorithm needed to model a particular target language, hence to estimate the probability of the $n+1$ word given a sequence of n previous words;

Standard monolingual neural language models take as input a list of symbols (e.g. words) $\mathbf{x}_{1:n} = [w_1, w_2, \ldots, w_n]$ where each symbol are converted to their respective embedding using the embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}_s|\times l}$ which give $\mathbf{W} \in \mathbb{R}^{n\times l}$, the words input matrix. This input matrix is then fed to a `LM` that outputs a hidden state, $\mathbf{h} \in \mathbb{R}^{l}$.

In some cases, like with the Transformer architectures, a list of n hidden states are generated, $\mathbf{H} \in \mathbb{R}^{n\times l}$. These n hidden states are then multiplied by the tied transposed embedding table in order to obtain a score for each word in the vocabulary according to the i-th hidden state; $\mathbf{H}\mathbf{E}^\top = \hat{\mathbf{P}} \in \mathbb{R}^{n\times|\mathcal{V}_s|}$. By performing the `argmax` operation on the columns, we thus obtain the sequence of estimated next words, $\hat{\mathbf{p}}_{2:n+1} = [\hat{w}_2, \hat{w}_3, \ldots, \hat{w}_{n+1}]$.

In this setting, we assume that we have access to a pre-trained monolingual neural language model such as Dai et al. (2019) that is able to generate natural language based on an embedding table $\mathbf{E}^s \in \mathbb{R}^{|\mathcal{V}_s|\times n}$ that is tied to the last layer of that same architecture as introduced in Section 2.3.1 which we define as $\mathbf{M} \in \mathbb{R}^{n\times|\mathcal{V}_s|} = \mathbf{E}^{s\top}$. This mapping layer $\mathbf{M}$ is used to obtain a probability distribution for the next word over the vocabulary given a history of symbols.

In this context, we define the pre-trained language model's **core component** as a function that takes as input a list of i word embeddings and outputs a list of i hidden representations; $\mathcal{C} : \mathbf{W} \in \mathbb{R}^{i\times n} \to \mathbf{H} \in \mathbb{R}^{i\times n}$. Usually, in order to retrieve the prediction of the next words in the sentence, we simply multiply $\mathbf{H}$ with the mapping layer $\mathbf{M}$ which gives us the following language model function $\mathcal{L} : \mathcal{C}(\mathbf{W})\mathbf{M} \to \mathbf{P}^s \in \mathbb{R}^{i\times|\mathcal{V}_s|}$. $\mathbf{P}^s$ constitutes the list of predictions of all the words in the sentence **in the source language**.

We thus propose to bypass the source mapping layer $\mathbf{M}$ and directly multiply $\mathbf{H}$ with a translation layer $\mathbf{T}_{s\to t}$ and the target embedding table $\mathbf{E}^t$ which will give us $\mathbf{P}^t \in \mathbb{R}^{i\times|\mathcal{V}_t|}$. Since $\mathbf{T}_{s\to t}$ has been trained so that $\mathbf{e}^t_i \approx \mathbf{e}^s_i\mathbf{T}_{s\to t}$, we can assume that $\mathbf{H}\mathbf{T}_{s\to t}\mathbf{E}^t$ will produce the list of most probable next words translated in the target language, $\mathbf{P}^t$.

This proposal offers a Zero-Shot [4] transferred language model $\mathcal{L}^t : \mathcal{C}(\mathbf{W})\mathbf{T}_{s\to t}\mathbf{E}^t$ that uses a core component of a pre-trained source language model $\mathcal{C}$, takes as input a list of embeddings in the source language $\mathbf{W}$ and generates a list of words[5] in the target language $\mathbf{P}^t$. Figure 3.1 depicts the overall architecture. It is important to note that this architecture is fully derivable and can thus be fine-tuned on the target language in order to obtain better results. This training scheme will indeed be studied during the experimentation.

[4]This setting is Zero-Shot in the sense that the we do not train or fine-tune the pre-trained language model

[5]We assume that we begin the sentence in the source language with the start token `<S>`. This way, we will in fact generate all the next words in the target sentence.

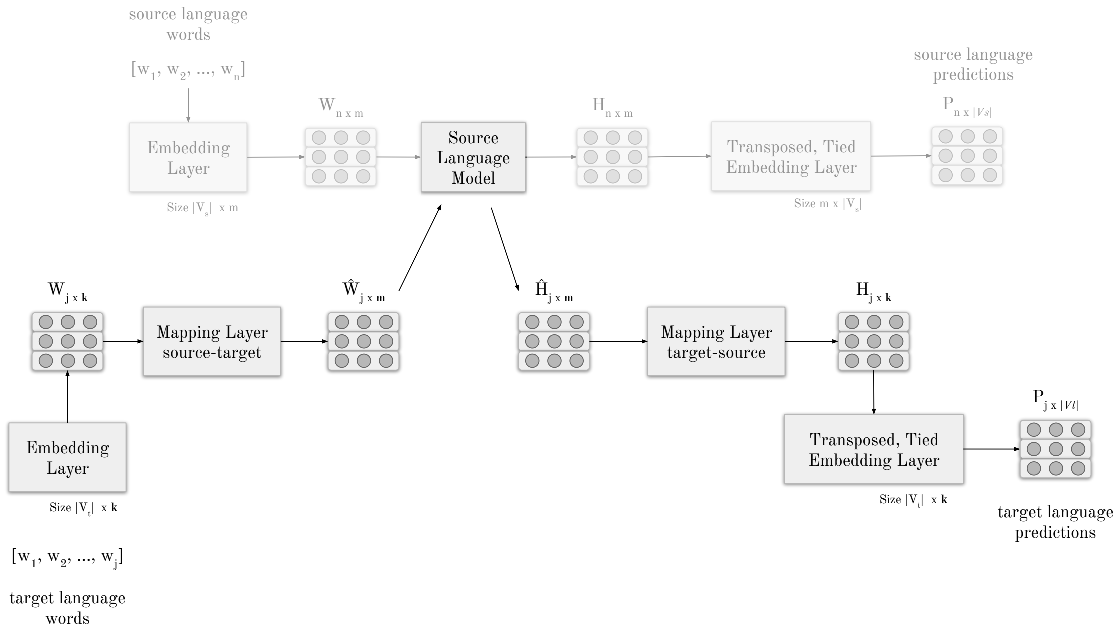


Figure 3.1 – The overall architecture of the proposed cross-lingual neural language model for textual generation. The gray area is a standard monolingual language modelling workflow. We thus propose to transfer a source monolingual language model towards the target language using two mapping layers, from source to target and target to source.

In order to evaluate our transferred architecture, we will proceed into an intrinsic and extrinsic evaluation. Indeed, in every transferred language, we can use the perplexity to evaluate the performance of our model. Since perplexity scores are inherently incommensurable between different languages, we will compare the performance of our model with robust baselines (Merity et al., 2017a) trained only on the target language's dataset as well as multilingual architectures such as XLM (Lample and Conneau, 2019). There are actually 2 datasets on which we intend to perform the evaluation, the Parallel Universal Dependencies dataset (Nivre et al., 2016) as well as the Europarl dataset (Koehn, 2005). To extrinsically evaluate the relevance of the generated text by our transferred model, we propose to manually analyze sentences in languages where the author is at ease, which are French, English, and Spanish.

Going Further

A fundamental problem lies in the proposed version of our model; since word order may differ from one language to another (e.g. from English to German), a simple word-to-word translation won't suffice in order to obtain the desired results in most target languages. Take for example the following English sentence: "`Erik is coming home on the train today.`". The word order in this case is characterized by the *place*, the *manner* and then the *time*. In German, it is the exact opposite: "`Erik kommt heute mit der Bahn nach Hause.`" which literally translates to "`Erik is coming today on the train home`" in English and barely makes sense. We thus proposed to improve our architecture by introducing a "positional aware layer". This layer could mitigate the word order issue that exhibits several language pairs and yield better results on the effective target language. The layer as depicted in Figure 3.2 will be trained to reorder words from a source language sentence given shuffled words, largely inspired by the work done by Vinyals et al. and Andrychowicz and Kurach.

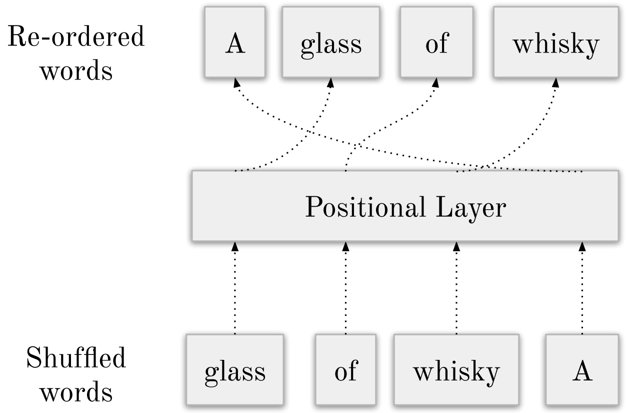


Figure 3.2 – A positional layer that learns how to properly reorder words in a sentence.

While the formal definition of the positional aware layer is not fully settled, we define the problem source embeddings reordering as follows;

1. Given a sequence of n shuffled source word embeddings $\mathbf{x}_1^s \dots \mathbf{x}_n^s$ with the associated correct ordering $y_1 \dots y_n$;
2. Use the Positional-Layer to find the best ordering given the shuffled sequence of word embeddings $\hat{y}_1 \dots \hat{y}_n$;
3. Record the permutations that have been predicted by the Positional-Layer for the re-ordering in the target language.

Another problem will arise when comes the time to transfer an English neural language model towards a morphologically rich language. The number of words used to express a given thought may differ from one language to another. Take for example this simple English sentence: `The boy plays with the girl`. Literally translated into Danish, this sentence has 4 words instead of 6: `Drengen leger med pigen`. The words "`Drengen`" and "`pigen`" are characterized by the suffix "`en`" which literally translates to "The boy" and "the girl". While a word-to-word translation from Danish to English may result in the following plausible sentence: "Boy plays with girl", it does not fully capture the fact that there is **a boy** that plays with **a girl**.

As stated in Ruder et al. (2018), words also combine in non-compositional ways to form multi-word expressions like "*kick the bucket*" or "*woke up*" where the meaning cannot be derived from standard word representations of their constituents. Multi-word expressions are still a huge challenge in the monolingual setting and have been barely studied with cross-lingual architecture. Since this problem is challenging and under-explored, we see a huge research potential into it.

3.2 Transferring Neural Language Models for Multimodal Conditional Textual Generation

In this proposal, we plan to tackle multimodal neural language modelling using an insurance dataset that is provided by a private partner. The dataset comprises email conversations between insurance brokers and underwriter agents. The idea here is to facilitate the work of the underwriters by suggesting or pre-generating an answer to the broker by leveraging contextual information.

These conversations contain textual data (dialogues) but may contain as well images, PDF forms or even scanned documents. Specific information related to the client's profile who is the subject of the exchange can even be leveraged to guide the language model as well as the company's internal business rules. For this project, we thus want to obtain a neural language model that can generate a response to the broker given the context that we have in hand;

1. The whole history of the conversation.
2. Were there any pictures/forms sent in the incoming message or required for the case?
3. What is the actual status of the broker's client?
4. Is there any specific business rules that need to be taken into account?

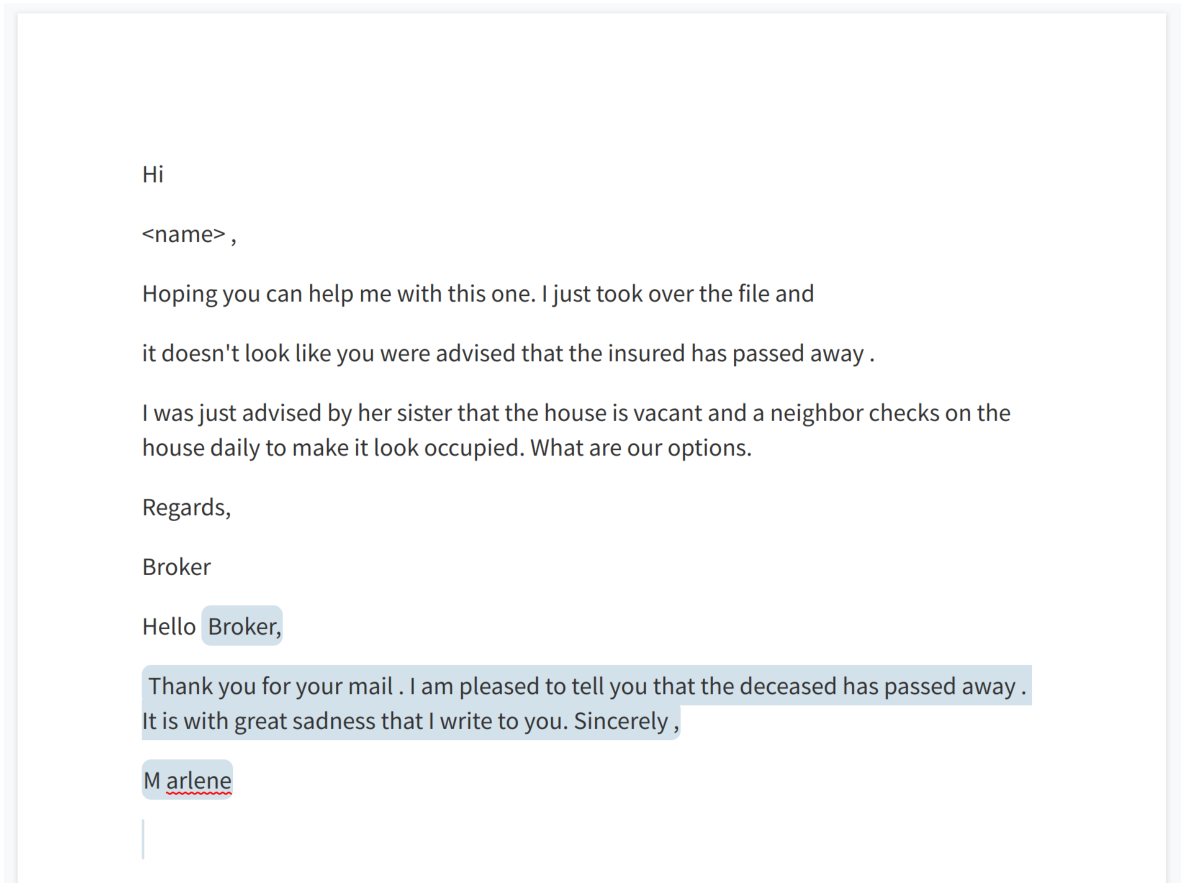
A typical email exchange is depicted in Appendix C. From this conversation, we can clearly see that besides the history of the discussion, a dialogue system that aims to generate a response to the broker will need more contextual information. The model should have 2 main abilities; answer a question from the broker and ask for specific information. The case in Appendix C demonstrates that the underlying language model will need to have access to the client's profile, to ask for missing information like the name and address of the client. It will also need to have access to the business rules of the company whereas if the dwelling becomes vacant, the broker needs to file up a Vacancy Questionnaire for the corresponding client.

In this context, we wish to explore conditional neural language generation. Indeed, we wish to generate the agent's response given the history of the exchange also leveraging the attachments of the different emails as well as the client's profile and specific business rules.

The dataset that we have in hand is composed of around 600,000 email exchanges, each comprising in average 3 images and 0.5 PDF form. Textual preprocessing needs to be done on the emails to extract and filter email headers, signatures, and replies.

3.2.1 Related Work

We show in Figure 3.3 that simply using a pre-trained instance of GPT-2 to generate an answer to the broker is not suitable for our needs[6]. Without the context (e.g. the client's profile and business rules associated to this specific case), the model is good for nothing in terms of generating a proper response impersonating the insurance company's underwriter. Of course, we could fine-tune GPT-2 on our conversation dataset but still contextual information won't be leveraged in order to generate a useful answer to the broker.



Hi

<name> ,

Hoping you can help me with this one. I just took over the file and

it doesn't look like you were advised that the insured has passed away .

I was just advised by her sister that the house is vacant and a neighbor checks on the house daily to make it look occupied. What are our options.

Regards,

Broker

Hello Broker,

Thank you for your mail . I am pleased to tell you that the deceased has passed away . It is with great sadness that I write to you. Sincerely ,

M arlene

Figure 3.3 – An example generation from the original GPT-2 model given a question asked by the broker. We can clearly see that without the context (client profile and business rules associated to this specific case), the model is good for nothing in terms of generating a proper response impersonating the insurance company's underwriter.

As discussed in Section 1.3.2, it is possible to obtain a representation vector for the history of the conversation using a language model. It is also possible to obtain a representation vector of an image using a convolutional neural network such as a ResNet (He et al., 2015) trained on ImageNet. The difficulty of this problem lies in obtaining the vectorial representation of a form (is it well filled or not), the client's profile (which takes the form of structured data), the business rules, and finally find a way to wisely combine all this information in order to guide the textual generation of a pre-trained language model.

[6]The original answer from the underwriter can be found in Appendix C

We introduced conditional neural language models in Section 2.4.2 that were mostly RNN-based models. These models are conditioned on different "modalities" such as text, images and structured data.

A line of research that is currently under-explored with the rise of the state-of-the-art transformer-based neural language model is indeed the **multimodal** aspect of such architecture; combining different types of inputs (images, structured data, audio, etc.) to generate textual utterances. The literature also refers to this field of research as *grounding natural language* and we thus wish to explore this field specifically towards transformer-based architectures.

Recently, a Transformer-based model has been proposed by Salesforce (Keskar et al., 2019), CTRL, which aims to control the generation of textual utterances. The mechanism employed relies on *control codes* that govern style, content, and task-specific behaviour. These control codes are basically special tokens that are prepended to training sequences of raw text. Still, this model cannot take into account different types of conditioning elements as we wish to do with our specific dataset and requires a huge amount of training data since it is composed of 1.6 billion parameters. Needless to say, that this architecture also requires substantial computing power.

Solutions proposed so far not only needs to modify the pre-trained architecture but often need a fine-tuning procedure or even a full training of the said architectures for conditional language modelling.

We thus propose to explore another form of conditioning that are often called **adapters** (Rebuffi et al., 2017; Houlsby et al., 2019) which reduce the computational resources needed for multimodal neural language modelling with conditioning elements never seen to date du to the nature of our dataset.

3.2.2 Research Proposal

It is clear to see from Figure 3.3 that we cannot use a pre-trained language model to generate a plausible answer. We either have to fine-tune it or condition the model in some way to generate meaningful answers to the broker.

This research avenue focuses on two main challenging aspects. At first, we need to find a way to encode the contextual information that is not in textual form. Indeed, actual pre-trained language models take as input a list of symbols or tokens which acts as the "textual conditioning". We thus need to find a way to encode images, PDF forms, client profile, and business rules in order to combine their respective representation with the history of the conversation.

Furthermore, we wish to minimize the needed resources in order to achieve a decent textual generation. We thus consider a pre-trained Transformer-based language model (Radford et al., 2019) as much as a black box as possible which will alleviate the need for fine-tuning or a complete training of the architecture.

Encoding Contextual Information

We hereby lay down three different types of contextual information as well as how we plan to encode them in order to be used within our conversational language model.

Images. As a baseline, we plan to use a convolutional neural network such as He et al. (2015) trained on the ImageNet dataset (Deng et al., 2009). With this pre-trained network, we can obtain a vector representation given an image. However, we would prefer to obtain the predicted class of the image amongst the 1,000 classes available in the architecture so that we can directly input this class (i.e. word) as a feature to the language model. Furthermore, as introduced in Section 2.4.2, such a pre-trained neural network has already been used to obtain an image's vectorial representation in order to generate the caption that describes it (You et al., 2016). The idea in our case is not to obtain an exhaustive description of the different images that may be attached to an email but rather obtain a bag of keywords that mostly summarizes all images (e.g. an image containing a cat could be described by the following keywords; cat, feline, grumpy, etc.).

Client Profile and PDF Forms. An insurance company's underwriters have to make sure that a client's profile is properly filled. This means that for every mandatory field that would have been omitted in his profile, we would simply encode the field name (e.g. executor's full name and address) as a bag of words to guide the language model into asking for these elements in the conversation. For example, if the client's first name is missing, it will be encoded as a special token as follows: `First_Name:Empty`. It will also be relevant to encode already collected information to provide more context to the language model; if the client's first name is Nicolas, it will thus be encoded as follows: `First_Name:Nicolas`. The same goes procedure applies to fields within PDF forms. This way of encoding different is from what has been introduced in Section 2.4.2 due to the nature of the model we wish to use, the Transformer-based language models.

Business Rules. Insurance company's processes are supported by many business rules that ensure a good cohesion between the different departments. An example of such a business rule can be drawn from the Appendix C: In order to set the dwelling vacant, a Vacancy Questionnaire must be filled out with associated photos for review. This information can indeed be used to guide a generative conversational agent to ask for this specific information given the context. Using such a knowledge base to **ground** a language model is further explored in Section 3.3.

Conditioning Pre-Trained Language Models for Generation

Dathathri et al. proposed an interesting way to control attributes of the generated language for a given Transformer-based language model architecture. Inspired by Plug & Play Generative Networks from Nguyen et al. (2016) used to generate images with different attributes, they introduce the Plug & Play Language Model for controllable text generation. We are particularly interested into this Plug and Play architecture since it circumvents the catastrophic forgetting that could happen to the pre-trained language models, as illustrated by Houlsby et al. (2019) and Wang et al. (2020).

Their architecture is composed of a pre-trained language model and one or more simple attribute classifiers that guide the generation without any further training of the language model (which can be really expensive). Their attributed classifiers actually only handle textual data. The main idea is that these simple attribute classifiers are used to shift the output distribution from the pre-trained model over the vocabulary. We borrowed Figure 3.4 from the original paper which properly illustrates the underlying process of steering the generation towards a specific topic using an attribute model.

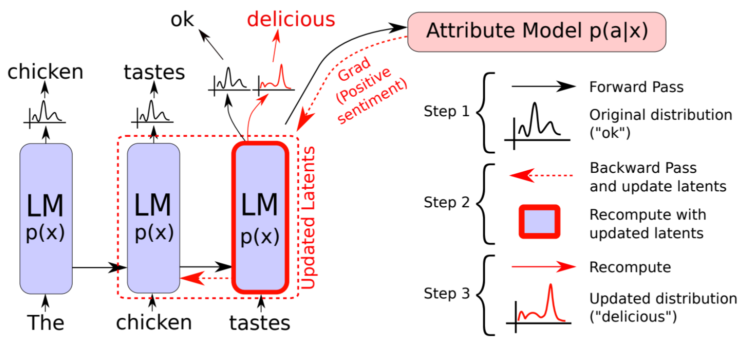

ok
delicious
Attribute Model p(a|x)
chicken
tastes
Grad (Positive sentiment)
LM p(x)
LM p(x)
LM p(x)
Updated Latents
The
chicken
tastes
Step 1
Forward Pass
Original distribution ("ok")
Step 2
Backward Pass and update latents
Recompute with updated latents
Step 3
Recompute
Updated distribution ("delicious")

Figure 3.4 – Illustration of a plug and play attribute model borrowed from the original paper (Dathathri et al., 2019).

These models are often called "plug and play" models since they only use the pre-trained architecture without the need for retraining it. The smaller attributes models such as a simple bag-of-word model or discriminative neural network are used to update the pre-trained model's latents using negative gradients.

More precisely, given an attribute $a =$ `positivity`, an attribute model is designed to increase the likelihood $p(x|a)$ of the passage x having the desired attribute a by updating the latents $\mathbf{H}$ of the Transformer-based language model, formulating the attribute model as follows; $p(a|\mathbf{H})$. The internal latent representations of the language model are updated using gradients from the attribute model, noted $\mathbf{H}_t = \mathbf{H}_t + \Delta\mathbf{H}_t$ where $\Delta\mathbf{H}_t$ are the gradients received from the attribute model. Their attribute model is thus formulated as follows: $p(a|\mathbf{H}_t + \Delta\mathbf{H}_t)$ [7].

As previously stated, this model has been used solely with textual attribute models. We thus propose to handle other types of modalities by encoding them in a vector representation used to shift the output distribution of the pre-trained language model. Based on the work of Dathathri et al., we plan to extend the formulation of attribute models to a vector space representation;

$$p(\mathbf{v}|\mathbf{H}_t + \Delta\mathbf{H}_t) \tag{3.1}$$

[7]Note here that they iteratively update $\Delta\mathbf{H}_t$ (around 3 to 10 times) in order to obtain the final conditioned output distribution over the vocabulary. At the initial time step, $\Delta\mathbf{H}_t = \mathbf{0}$, resulting in the instance of the original attribute model $p(a|\mathbf{H})$

such that $\mathbf{v}$ is a vector representation obtained by the methods exposed in Section 3.2.2. This results in a really interesting architecture that worth exploring further in order to design attribute models that can handle structured data such as images, the client's profile or the business rules which our dataset inherently exhibits.

However, we introduced in Section 3.2.2 several types of contextual information that may need to be taken into account for the language model in order to provide relevant answers (or even questions!) to the corresponding broker, yielding many possible representations of $\mathbf{v}$.

This is why we want to explore conversation planning which could potentially help the conversational agent to assign a ranking to the different topics that must be addressed based on the contextual information. As introduced in Section 2.4.2, Kiddon et al. (2016) and Moryossef et al. (2019) specifically tackle this issue and we wish to explore how these architectures could be incorporated into a transformer-based neural language model.

Suppose here that we have two vector representations contextualizing the conversation; the client's profile ($\mathbf{v}_1$) and an image ($\mathbf{v}_2$). We thus propose the following vectorial attribute model;

$$p(\mathbf{v}_i|\mathbf{H}_t, [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{t-1}]) \tag{3.2}$$

where $\mathbf{v}_i$ is the i-th attribute vector in the set of possible attribute vectors $\mathbf{V}$, $\mathbf{H}_t$ encodes the history of the whole conversation and $[\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{t-1}]$ contains the history of attribute vector representations used so far in the generation [8]. The attribute vector that has the highest probability could then be used to update the transformer's latents at time step t;

$$\mathbf{v}_t = \underset{\mathbf{v}_i \in \mathbf{V}}{\operatorname{argmax}} \left(p\left(\mathbf{v}_i|\mathbf{H}_t, [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{t-1}]\right) \right) \tag{3.3}$$

3.3 Grounding Multilingual Neural Language Models Using Structured Data

In this avenue, we move away from language generation but we continue to study pre-trained transformer-based language models. We specifically study the multilingual version of BERT (Devlin et al., 2018), mBERT, which proved to perform well on a wide variety of sequence tagging and text classification tasks (Pires et al., 2019; Lample and Conneau, 2019).

To this end, we wish to evaluate the global consistency of mBERT, per language, using a new cross-lingual analogy dataset [9] extracted from Wikidata [10]. We thus want to perform multilingual grounding of mBERT on these extracted analogies in order to understand the model's learning dynamics.

[8]We could in fact only use a truncated list of attributes.

[9]A special thanks to Anders Sandholm for the extraction.

[10]`https://www.wikidata.org/wiki/Wikidata:Main_Page`

3.3.1 Related Work

Distributional (Mikolov et al., 2013b; Pennington et al., 2014) and contextualize (Peters et al., 2018; Devlin et al., 2018) word representations are obtained from a model trained in an unsupervised manner on a large textual corpus.

Also known as "semantic specialization", language grounding's goal is to provide language word representations that complies to a set of semantic rules. These rules are usually drawn from a different normalized data source than the one the model has been trained on such as Wikidata. As an example, one may want to specialize word representations to capture synonymy-antonymy (Mrksic et al., 2017), phenomena poorly captured by a vanilla distributional model such as skip-gram.

Neural language grounding has been studied since the rise of distributional representations induced from a neural architecture (Mikolov et al., 2013b; Pennington et al., 2014). In the case of distributional models, semantic specialization methods usually fall into two main categories:

1. Those that modifies the distributional model in order to handle both distributional knowledge and lexical information (Liu et al., 2015; Schwartz et al., 2015; Ono et al., 2015);
2. Those that inject lexical information within trained representations induced from the distributional models (Rothe and Schütze, 2015; Mrksic et al., 2016, 2017; Vulic et al., 2017; Vulić and Mrkšić, 2018).

These models are usually evaluated in an analogy detection task (detailed in Section 1.2), synonymy-antonymy and in hypernymy-hyponymy detection setting, just to name a few. To date, specialization methods have been studied in a cross-lingual setting where the original distributional models were monolingual (Ponti et al., 2018, 2019) and the specialization was done across languages.

In this proposal, we plan to inject cross-lingual lexical information within a pre-trained cross-lingual transformer-based language model, mBERT (Devlin et al., 2018), using a new specially curated multilingual analogy dataset. This proposal differs from previous work in two main aspects;

1. Using pre-trained cross-lingual contextualized representations;
2. Using a cross-lingual analogy dataset extended with entity metadata.

We thus detail next these two specific aspects which we think make this research avenue really appealing.

3.3.2 Research Proposal

As previously mentioned, in this proposal we wish to study the learning dynamics of mBERT, the multilingual version of BERT (Devlin et al., 2018) using a new cross-lingual analogy dataset. We propose a version of an analogically specialized mBERT as well as a baseline using multilingual Fasttext's embeddings [11] in order to compare our proposed architecture.

Dataset

Using the Wikidata knowledge base [12], we wish to extract relevant analogies in order to train and validate pre-trained word or phrase representations.

In Wikidata, entities are identified by a unique Q-Code and properties (P) characterize links between a pair of entities. For example, *Paris* (Q90) is related to *France* (Q142) via the property *Capital* (P36). Similarly, *Berlin* (Q64) is related to *Germany* (Q183) via the same property (P36). This allows us to create an analogy in that the **source** entity *Paris* should relate to the **target** entity *France* as *Berlin* relates to *Germany.*

In general, for a given property P_i, we would extract an analogy $(Q_1^s, Q_1^t, Q_2^s, Q_2^t)$ if both triples (Q_1^s, P_i, Q_1^t) and (Q_2^s, P_i, Q_2^t) are in the Wikidata knowledge base.

Wikidata offers its knowledge base in various languages. We thus propose to extract a large set of analogies across languages which constitutes, to the best of our knowledge, the first cross-lingual analogy dataset. We present in Table 3.1 the property *is part of* that characterizes two entities of type *unit of time* in different languages that compose Wikidata.

Lang.	Source	Target
da	time → døgn	døgn → uge
de	Stunde → Tag	Tag → Woche
en	hour → day	day → week
es	hora → dia	dia → semana
fi	tunti → vuorokauski	vuorokauski → viiko
fr	heure → jour	jour → semaine
it	ora → giorno	giorno → settimana
nl	uur → dag	dag → week
pl	godzina → dzień	dzień → tydzień
pt	hora → dia	dia → semana
sv	timme → dygn	dygn → vecka

Table 3.1 – Example pairs of the relation *unit of time* (Q_1) → *is part of* → *unit of time* (Q_2) in the different languages that compose our dataset. Each row contains a source (S) and a target pair (T).

In some cases, the entities are composed of only one word (has illustrated in Table 3.1) which leaves a contextual model like BERT with too few information in order to generate a

[11] `https://fasttext.cc/docs/en/crawl-vectors.html`
[12] `https://www.wikidata.org/`

meaningful representation for a given entity. To circumvent this, we propose to use longer aliases from Wikidata to replace the words before the vectorization.

Consider the same entity *unit of time* and characterized by the property *is part of*. We can then instantiate this analogy in French, whereas *jour* → *is part of* → *semaine* as to *semaine* → *is part of* → *mois*. BERT have little information in order to deduce this analogy. We can then use the description of the entities provided by Wikidata [13] which describes a *jour* and expand the entities as *jour, unité de temps de 24h*, *semaine, unité de temps de 7 jours* and *mois, unité de temps*, yielding the following augmented analogy;

jour, ***unité de temps de 24h*** → *is part of* → *semaine,* ***unité de temps de 7 jours***
as to
semaine, ***unité de temps de 7 jours*** → *is part of* → *mois,* ***unité de temps***

The new multilingual analogy dataset that we propose here is composed of 11 languages and a total of 78 thousand pairs.

Grounding mBERT

At first, we propose to show that multilingual BERT – and similar pretrained language encoders representing multiple languages in the same vector space - are comparatively worse at capturing relationships between words that are further apart in languages with less training data.

To validate our hypothesis, we will use the analogy dataset previously introduced to perform analogy detection. Analogy detection can be seen as a retrieval task such that given two pairs (Q_1^s, Q_1^t) and (Q_2^s, Q_2^t), we want to retrieve Q_2^s by using Q_1^s, Q_1^t and Q_2^t.

We can obtain the vectorial representation of an entity Q by simply feeding the tokens as an input;

$$\mathbf{v} = \text{mBERT}([t_1, t_2, \dots, t_n]) \tag{3.4}$$

where $\mathbf{v} \in \mathbb{R}^d$ and $[t_1, t_2, \dots, t_n]$ would be the tokenized representation of an entity, like *Barack Obama, the 44th president of the United States* for example.

Given the corresponding vector representations of each entity in our dataset, $\mathbf{V} \in \mathbb{R}^{m \times d}$, we can use the *offset trick* proposed by Mikolov et al. to get an estimation of the vectorial representation of Q_2^s, $\mathbf{v}_{Q_2^s}$;

$$\hat{\mathbf{v}}_{Q_2^s} = \mathbf{v}_{Q_1^s} - \mathbf{v}_{Q_1^t} + \mathbf{v}_{Q_2^t}. \tag{3.5}$$

We thus propose to implement a four-way Siamese BERT architecture based on Sentence-BERT (Reimers and Gurevych, 2019) to ground mBERT on Wikidata. Since every analogy

[13] https://www.wikidata.org/wiki/Help:Description/fr

in the dataset is offered in 11 different languages, we can then assess the multilinguality of mBERT by fine-tuning it on the whole dataset and evaluating it per language.

We can train this proposed architecture by using an objective function that tries to bring together the estimated and the true vectorial representation of Q_2^s using the cosine distance;

$$\mathcal{J}(\Theta) = \frac{1}{N}\sum_{n=1}^{N} \text{cosine\_distance}(\hat{\mathbf{v}}_n, \mathbf{v}_n) \tag{3.6}$$

where Θ are the parameters of mBERT, $\mathbf{v}_n = \mathbf{v}_{Q_2^s}$ and given we have N analogies in our training set. This first objective function is said to be the *explicit* one.

We also propose a more *contrastive* objective function, leveraging negative samples (see Equation 3.7). The idea is to use a cross-entropy loss to distinguish between correct pairs of entities and randomly constructed ones;

$$\mathcal{J}(\Theta) = \frac{1}{N}\sum_{n=1}^{N}\left(\log\left(1 + e^{-cos(\hat{\mathbf{v}}_n, \mathbf{v}_n)}\right) + \sum_{j=1}^{k}\log\left(1 + e^{cos(\hat{\mathbf{v}}_n, \mathbf{v}_j)}\right)\right) \tag{3.7}$$

where the k negative samples are drawn randomly and *cos* is simply the cosine similarity. However, in order to make the contrastive loss converge more rapidly, we may have to test different sampling functions based on type constraints.

For instance, given the following analogy of languages and their corresponding writing systems; *runes* $\rightarrow$ *Old Norse* as *cuneiform* $\rightarrow$ *Akkadian.* A randomly sampled negative example could be *runes* $\rightarrow$ *Old Norse* as *Britney Spears* $\rightarrow$ *Akkadian.* However, the property type of *cuneiform* and *Akkadian* is P407 (language of work name) while *Britney Spears* of type Q5 (human) is not characterized by this property. It would make more sense to draw an entity of the same type (Q8192, writing systems) that allows this property type such as *kanji* for example.

To evaluate the performance of our proposed architecture, we will rely on the accuracy metric. In order to compute it, we assume that we have the vectorial representation of every entity in our dataset, $\mathbf{V} \in \mathbb{R}^{m \times d}$; we can compute the accuracy as follows;

$$\begin{aligned} \mathbf{s} &\leftarrow \hat{\mathbf{v}}_{Q_2^s}\tilde{\mathbf{V}}^{\top} \\ p &\leftarrow \text{argmax}(\mathbf{s}) = h(Q_2^s) \\ \text{accuracy} &\leftarrow \frac{1}{N}\sum_{n=1}^{N} p_n \end{aligned} \tag{3.8}$$

where $\tilde{\mathbf{V}} = \mathbf{V} \setminus \left\{\mathbf{v}_{Q_1^s}, \mathbf{v}_{Q_1^t}, \mathbf{v}_{Q_2^t}\right\}$ and $h(\cdot)$ returns the index of a given entity in matrix $\mathbf{V}$.

In order to compare our proposed model, we will use Fasttext cross-lingual word embeddings as a baseline on the same analogy task. Since most of our analogies are composed of multi-word expressions, we propose to take the mean of Fasttext's embeddings for a given entity as follows;

$$\mathbf{v}_Q = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \mathbf{w}_i \tag{3.9}$$

where $|Q|$ is the number of words composed in entity Q and $\mathbf{w}_i$ is the Fasttext's embedding of the i-th word in the entity Q.

Finally, we wish to evaluate extrinsically our proposed mBERT grounded architecture on the XTREME dataset (Hu et al., 2020) and on the Universal Dependencies dataset within a Named Entity Recognition sequence tagging task.

Conclusion

In this research proposal, we introduced different ways to leverage pre-trained neural language models with minimal resources and effort. We proposed three research avenues with a strong innovative potential. We have a special attention towards scenarios with limited resources, like underrepresented languages. We hope that the proposed architectures will stimulate the scientific community towards this direction.

By transferring large pre-trained neural language model, we avoid the cost associated with the training procedure (dataset and computational power) which can cost several thousand US dollars. These substantial savings will facilitate the use of these high-energy models.

It goes without saying that there is an environmental consideration that is too often neglected when it comes to training such powerful models. Indeed, as presented by Strubell et al., the complete training of one of the previous models (with research hyperparameters) produces as much CO_2 emissions as the life of 6 conventional cars. This is enormous. We therefore have every interest to take advantage of the training that has been done previously for the benefit of other languages hence transferring this learning to other contexts.

Appendix A

Long-Short-Term-Memory Networks

This recurrent cell comprises 4 sub-networks and 3 gates called the forget, input and output gates. The internal computations of the cell are exposed in Equation A.1

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{ii}\mathbf{x}_t + \mathbf{b}_{ii} + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_{hi}) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_{if}\mathbf{x}_t + \mathbf{b}_{if} + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_{hf}) \\
\mathbf{g}_t &= \mathtt{tanh}(\mathbf{W}_{ig}\mathbf{x}_t + \mathbf{b}_{ig} + \mathbf{W}_{hg}\mathbf{h}_{t-1} + \mathbf{b}_{hg}) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{io}\mathbf{x}_t + \mathbf{b}_{io} + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_{ho}) \\
\mathbf{c}_t &= \mathbf{f}_t \times \mathbf{c}_{t-1} + \mathbf{i}_t \times \mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \times \mathtt{tanh}(\mathbf{c}_\mathtt{t})
\end{aligned}
\tag{A.1}
$$

It is important to note that the weights $\mathbf{W}_{i\_} \in \mathbb{R}^{j \times l}$ and $\mathbf{W}_{h\_} \in \mathbb{R}^{l \times l}$ are related to the input and the hidden state respectively and $\mathbf{W}_{\_i}$, $\mathbf{W}_{\_f}$, $\mathbf{W}_{\_g}$, $\mathbf{W}_{\_o}$ are associated to the input gate, forget gate, update layer and output gate respectively. These are all sets of different parameters of the recurrent cell. The same pattern also applies to the biases. We depicted there LSTM architecture in Figure A.1 which is highly inspired by Christopher Olah's blog[0]. Cho et al. introduced a variant of this model, the Gated Recurrent Unit (GRU) reducing the number of sub-networks to three.

[0] `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

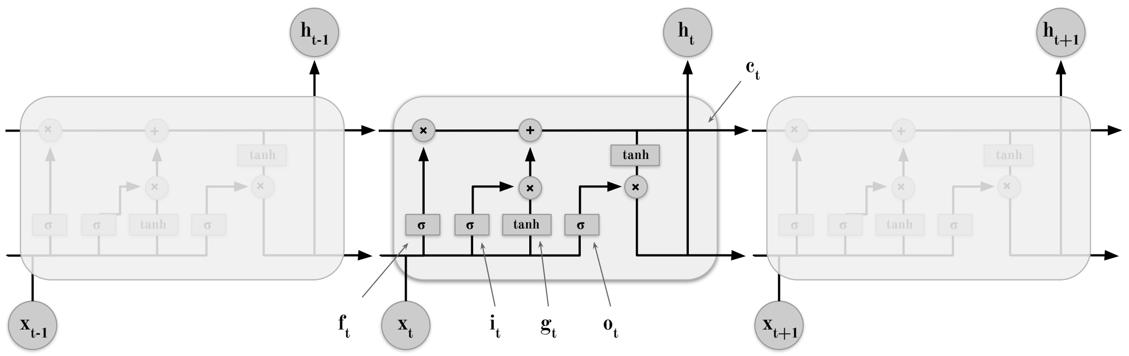


Figure A.1 – A Long-Short-Term-Memory Neural Network. The four boxes in the bottom of the figure represents the four neural layers. The circles are pointwise operations. We also depicted where the different outputs ($\mathbf{f}_t$, $\mathbf{i}_t$, $\mathbf{g}_t$, and $\mathbf{o}_t$) as well as the cell state $\mathbf{c}_t$ stands in the architecture.

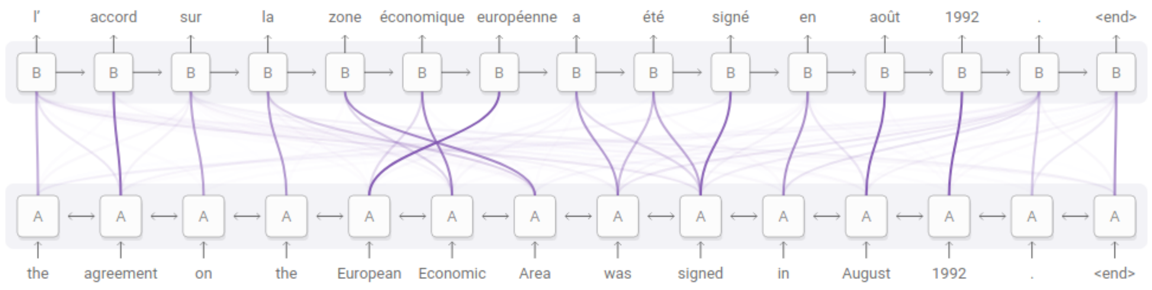


Figure A.2 – The behaviour of an attention mechanism in the machine translation task. This figure is taken from the really interesting post [1] on augmented RNNs by Chris Olah and Shan Carter (Olah and Carter, 2016) which is under the licence CC BY `https://creativecommons.org/licenses/by/2.0/`.

Appendix B

Language Models Training Details

When comes the time to concretely train a neural language model, there are lots of hyper-parameters and methods to consider. These methods are more often than not poorly documented in conference papers and we will hereby explicitly expose any internal parts of the training details of a neural language model.

We conclude this appendix by presenting different hyperparameters that can be tuned in order to produce coherent generated text.

The training loop

Language modelling is exactly the same as a classification problem; given some input $\mathbf{x}$, what is the probability of obtaining class y_i, the next word. In this section, we assume that the number of classes is the number of symbols in the vocabulary, $|\mathcal{V}|$, and that a language model is trained with Maximum Likelihood Estimation (MLE). MLE is usually estimated with the Cross-Entropy loss, thus obtaining the probability of the next symbol y_i as $P(y_i|\mathbf{x})$ using softmax function on the output of the network.

It may seem obvious that, in this classification setting, $\mathbf{x}_{1:k}$ is a sequence of k previous symbols and y_{k+1} is the next symbol to predict. Our first intuition to train such a language model could be to randomly sample a sequence of $k+1$ symbols in our corpus and use k symbols to predict the next one.

Merity et al. (2017a), which is the state-of-the-art RNN language model, did not exactly proceed that way. It only uses a sliding window of size k over the corpus to predict the $k+1$ symbol (word or character in their case). This gives a considerable boost to an RNN architecture, maintaining a coherent state throughout the training, comparatively to randomly sampling sequences. One important thing to note in their architecture is that the size of the window (or history) k changes at each batch of examples. Given the predicted symbol $\hat{y}$, the back-propagation algorithm is triggered and the optimizer then updates the weights accordingly to make a better prediction on the next "sampled" batch of examples. The impact of varying the size of k is discussed in the next section.

The importance of the window size k

When training a feed-forward language model, k is fixed since the number of input values need to be fixed. It is important to note that every beginning of documents (sentence, paragraph, etc.) will need to be padded by adding a special symbol at the beginning and at the end in order to be able to begin and terminate the textual generation.

Varying the window size k makes more sense when using RNN-based or Transformer-based language models; they are agnostic to the length of the sequence. As stated in section 2.3.1, an RNN-based language model can take as input a sequence of symbols of arbitrary length to generate the next symbol, from 1 to n where n can get pretty large. It thus removes the need to pad the sequences as with n-gram or feed-forward language models. Passing sequences of the same lengths to RNN-based or Transformer-based language models will lead to overfitting, making the models dependent of this fixed length k. The variation of the length k is really important, even more when comes the time to *generate* sequences.

Teacher Forcing

Usual language models are trained using Teacher Forcing (Goodfellow et al., 2016, Chapter 10). In a language modelling perspective, suppose we have a sequence of symbols $Y = (y_1, y_2, \ldots, y_n)$ that a language model needs to generate. This language model will start with the begin of sequence `<BOS>` token to predict the first symbol y_1, yielding $\hat{y}_1$. To predict the next symbol y_2, the model will not use its previous prediction $\hat{y}_1$ but the real previous symbol as input, y_1. The same routine goes on to predict y_3 using the previous real symbols y_1 and y_2 and so on until the end of the sequence.

This approach yields faster convergence during training but may lead the language model to drift during generation (or the test phase). Indeed, during generation, the language model is only able to use its previous prediction (auto-regressive modelling) to predict the next word. In the next section, we introduce Scheduled Sampling, a technique that mitigates this generation problem.

Scheduled Sampling

Scheduled Sampling is the process of randomly using the previous prediction of the model $\hat{y}_{i-1}$ with a probability p during training as input the of the next step ($w_i = \hat{y}_{i-1}$) to predict the next symbol y_i instead of using the real input symbol w_i from the corpus. The probability p of using the previous real symbol y_{i-1} or not is modelled with an inverse sigmoid function as illustrated in Figure B.1. This inverse sigmoid decay depends on the training step, as propose by Bengio et al. (2015) yield a higher probability to use previously predicted symbol $\hat{y}_{i-1}$ as the training progresses.

In the beginning of the training loop, we are mostly in a Teacher Forcing scheme (see B). Then the model gradually uses more and more its own prediction to predict the next word. This approach may offer better generalization performance on natural language generation since

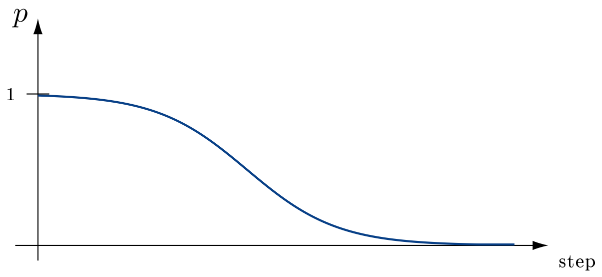


Figure B.1 – The probability of using y_{i-1} instead of $\hat{y}_{i-1}$ to predict y_i over time.

the model gradually learns to rely on its previously predicted symbols but it has not been thoroughly compared against Teacher Forcing.

B.0.1 Hyperparameters

Neural language models have several hyperparameters that are common to most neural network architectures (learning rate, batch size, etc.) that we won't discuss here. These hyperparameters comes into play during the training of a language model. Most of the hyperparameters that are of interest and we are going to discuss are the ones at generation time.

Decoding Algorithm

During generation, one can either do greedy decoding (taking the next most probable symbol) or sample the distribution (sampled generation) estimated by the language model over the vocabulary. Greedy decoding is deterministic while sampling is stochastic (it highly depends on the random seed).

Beam Search

Beam Search is an algorithm that optimizes the overall generation. It is highly used in machine translation to maximize the translation of a particular language. The hyperparameter to tune here is the size of the beam, ranging from 1 to 10. The beam size dictates the number of possible paths to consider during the generation. Koehn and Knowles laid a good overview of the beam size to use in the machine translation task claiming that a narrow beam (from 4 to 30) yields better results. However, choosing the right beam size is still based on a rule of thumb.

Temperature

Temperature scaling is a hyperparameter that is easy to tune and impacts only during sampled generation. Temperature is the coefficient by which the logits softmax function is applied. High temperature (> 1) flattens the distribution which will yield more diversity during sampling. Lower temperature (< 1) makes the probability distribution over the vocabulary more peaky, yielding a more conservative generation.

Top-K

Another hyperparameter that is highly tuned during generation is the number of symbols considered while sampling the distribution. Indeed, instead of considering the whole vocabulary while sampling, an approach that yields better results at generation is to consider only the Top-K symbols. K is usually set to 40 according to (Dai et al., 2019; Radford, 2018) for the best results on language generation.

Top-P

Instead of limiting the sampling to the K top symbols, Zellers et al. proposed to use a percentage of the distribution, P. In their paper, they used $P = 0.96$ which corresponds to using 96% of cumulative probabilities during sampling the vocabulary. However, the motivation behind this technique as well as the value chosen is not stated at all and they do not provide an order of magnitude on the number of symbols considered during sampling at each time step.

Appendix C

Typical Underwriter-Broker discussion

Back to Section 3.2

Hi `<name>`,
Hoping you can help me with this one. I just took over the file and **it doesn't look like you were advised that the insured has passed away**. I was just advised by her sister that the house is vacant and a neighbor checks on the house daily to make it look occupied. What are our options.
Regards,
Broker

=======

Hi `<name>`,
Thank you for your email.
We have been made aware that the insured has passed, but were under the impression that the sister was residing in the dwelling. **Can you advise the date the dwelling went vacant? We will require a Vacancy Questionnaire be filled out, and front and back photos of the dwelling submitted for review**.
Until that information is received and reviewed, there is currently no coverage on this dwelling.
Furthermore, what are the future intentions of this risk? Is it set to be sold?
Please advise on the above, than you.
Kind regards,
Insurance Company

=======

Hi,
I just spoke with the insured and clarified our conversation from last week.

She lives in the house 3 weeks on and 3 weeks off. For the 3 weeks that she is not in the home a neighbor looks after it.
The future intentions of the home are for it to be sold. However, they have to wait for probate to go through for the insured's husband who pre-deceased her, then insured's probate to go through before they can sell the house.
Please advise if the vacancy questionnaire and photos are still required.
Thank you,
Broker

=======

Hi `<name>`!
As per the above insured, **can you provide us with the executor's full name and address for our files?**
Also, would we need to amend the mailing address to that of the executor's address? Thank you!
Kind regards,
Insurance Company

=======

Hi,
The executor's name is `<name>` and her mailing address is `<address>`.
Please amend on the policy as well.
Regards,
Broker

Back to Section 3.2

Appendix D

Technical Details

In this section, we provide a methodology applicable to any Machine Learning project that will help us deliver highly reproducible experiments conducted during this research project (Garneau et al., 2019). We begin by introducing the programming language and associated libraries that are going to be used to implement the proposed architectures. We then propose a way to host the datasets that are going to be used off the shelves as well as those that are going to be derived from them for the specific settings. A tool that handles the download and the upload of the different dataset will also be introduced. Since another very important thing to consider when running experiments is to log them all, we hereby propose to automate the logging of the experimentations as well as the gathering of the results. These steps considerably facilitate the automatic generation of tables and graphs, hence removes human manipulations and possible errors. Finally, we recommend using a 100 % reproducible environment to run the experiments, hence to *Dockerize* the whole project (Cito and Gall, 2016; Hartmann et al., 2019).

D.1 Programming Language and Libraries

To implement the different introduced architectures, we will use the Python programming language. This programming language offers various data science libraries such as scikit-learn (Pedregosa et al., 2011) and numpy (van der Walt et al., 2011).

Pytorch (Paszke et al., 2019) is the neural network package that will be used to develop the different models previously introduced. Poutyne (Paradis, 2018) will be used to design the different training loops needed to optimize the neural network models. The HuggingFace (Wolf et al., 2019) library will be used to obtain the pre-trained implementations of the Transformer Models.

D.2 Fully Reproducible Methodology

Reproducing the model and the results of an original paper can be quite a hassle. This is why we present here a fully reproducible methodology so that the outcome of the research project can easily be used by the whole community.

D.2.1 Data Version Control

Properly handling the benchmark dataset(s) is often an underestimated point. In an iterative and collaborative setting, it is important to efficiently host *and* version the data. We thus recommend a tool designed to handle those two elements flawlessly; Data Version Control (DVC). Similar to standard Version Control Systems (VCS) like Git[1], DVC tracks the different state of the dataset during development as well as in between the processing steps before obtaining the final results of the model. We will be using the Python DVC library [2] with Amazon S3 as the remote repository needed to host our different datasets.

D.2.2 Experimentation Logging

When doing research, it is easy to enter the experiment's hurry loop; as soon as we have an idea, we code it and we launch our main script without committing the modifications. Grossly keeping track of an architecture and its corresponding results in our head or a spreadsheet is good for nothing when it comes to the time to retrieve and analyze past experiments. We thus propose to automate the process of logging as well as retrieving the results of every experiment in order to reduce the risk of losing experiment information. To this end, we used a flexible yet emergent tool that beautifully solves this problem; MLflow[3,4].

MLflow provides a model agnostic Python API that lets you track not only the results of a given configuration but also the associated source code, the dataset used, and much more. We also believe it is vital to use such a framework for any scientific team doing serious research to reduce the overhead and stress of manually logging and keeping the information about experimentations, especially considering the low effort it requires to setup.

D.2.3 Dockerization

Docker[5] is a software that provides an abstraction of the system libraries, tools, and runtime. A Docker Container is essentially a lightweight executable package that can run on every[6] environment. Providing a Docker Image that contains all the necessary libraries and source code is a mandatory step in order to provide fully reproducible research.

[1]`https://git-scm.com/`
[2]`https://dvc.org`
[3]`https://mlflow.org`
[4]One can find numerous alternatives such as Sacred, Comet.ml or Weights and Biases for example.
[5]`https://www.docker.com/`
[6]As long as the environment provides the necessary hardware specifications.

Bibliography

Rami Al-Rfou', Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. *CoRR*, abs/1808.04444, 2018.

Marcin Andrychowicz and Karol Kurach. Learning efficient algorithms with hierarchical attentive memory. *ArXiv*, abs/1602.03218, 2017.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings. In *ACL*, 2018.

Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. On the cross-lingual transferability of monolingual representations. *ArXiv*, abs/1910.11856, 2019.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Junwei Bao, Duyu Tang, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. Text generation from tables. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27:311–320, 2019.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *NIPS*, 2015.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, March 2003. ISSN 1532-4435.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH*, 2013.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *ArXiv*, abs/1406.1078, 2014.

J. Cito and H. C. Gall. Using docker containers to improve reproducibility in software engineering research. In *IEEE/ACM*, pages 906–907, 2016.

Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. *ArXiv*, abs/1710.04087, 2017.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric C. Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. *ArXiv*, abs/1912.02164, 2019.

Wietse de Vries, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. Bertje: A dutch bert model. *ArXiv*, abs/1912.09582, 2019.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

Georgiana Dinu and Marco Baroni. Improving zero-shot learning by mitigating the hubness problem. *ArXiv*, 1412.6568, 2014.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

Jessica Ficler and Yoav Goldberg. Controlling linguistic style aspects in neural language generation. *arXiv preprint arXiv:1707.02633*, 2017.

Nicolas Garneau, Mathieu Godbout, David Beauchemin, Audrey Durand, and Luc Lamontagne. A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings: Making the method robustly reproducible as well. *ArXiv*, abs/1912.01706, 2019.

Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2013.

Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

Chengyue Gong, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. Frage: Frequency-agnostic word representation. *ArXiv*, abs/1809.06858, 2018.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. Incorporating copying mechanism in sequence-to-sequence learning. *ArXiv*, abs/1603.06393, 2016.

Mareike Hartmann, Yova Kementchedjhieva, and Anders Søgaard. Comparing unsupervised word translation methods step by step. In *NeurIPS 2019*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9: 1735–1780, 1997.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *ICML*, 2019.

Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalization. *ArXiv*, abs/2003.11080, 2020.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Dan Jurafsky and James H. Martin. N-gram language models. In *Speech and Language Processing*, 2018.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *ArXiv*, abs/1909.05858, 2019.

Chloé Kiddon, Luke S. Zettlemoyer, and Yejin Choi. Globally coherent text generation with neural checklist models. In *EMNLP*, 2016.

Alexandre Klementiev, Ivan Titov, and Binod Bhattarai. Inducing crosslingual distributed representations of words. In *COLING*, 2012.

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. 2005.

Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. In *NMT@ACL*, 2017.

Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *CoRR*, abs/1901.07291, 2019.

Rémi Lebret, David Grangier, and Michael Auli. Neural text generation from structured data with application to the biography domain. In *EMNLP*, 2016.

Jiwei Li, Will Monroe, Tianlin Shi, Alan Ritter, and Daniel Jurafsky. Adversarial learning for neural dialogue generation. In *EMNLP*, 2017.

Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

Chia-Wei Liu, Ryan Lowe, Iulian V Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *arXiv preprint arXiv:1603.08023*, 2016.

Fenglin Liu, Yuanxin Liu, Xuancheng Ren, Kai Lei, and Xu Sun. Aligning visual regions and textual concepts: Learning fine-grained image representations for image captioning. *ArXiv*, abs/1905.06139, 2019a.

Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *CoRR*, abs/1801.10198, 2018.

Quan Liu, Hui Jiang, Si Wei, Zhen-Hua Ling, and Yu Hu. Learning semantic word embeddings based on ordinal knowledge constraints. In *ACL*, 2015.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. 2019b.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. 2008.

Mitchell P. Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *HLT*, 1994.

Louis Martin, Benjamin Muller, Pedro Javier Ortiz Su'arez, Yoann Dupont, Laurent Romary, 'Eric Villemonte de la Clergerie, Djamé Seddah, and Benoît Sagot. Camembert: a tasty french language model. *ArXiv*, abs/1911.03894, 2019.

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *CoRR*, abs/1707.05589, 2018.

Gábor Melis, Tomás Kociský, and Phil Blunsom. Mogrifier lstm. *ArXiv*, abs/1909.01792, 2020.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *CoRR*, abs/1708.02182, 2017a.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *CoRR*, abs/1609.07843, 2017b.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An analysis of neural language modeling at multiple scales. *CoRR*, abs/1803.08240, 2018.

Tomas Mikolov. Statistical language models based on neural networks. 2012.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013b.

Amit Moryossef, Yoav Goldberg, and Ido Dagan. Step-by-step: Separating planning from realization in neural data-to-text generation. In *NAACL-HLT*, 2019.

Nikola Mrksic, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gaić, Lina Maria Rojas-Barahona, Pei hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. Counter-fitting word vectors to linguistic constraints. In *HLT-NAACL*, 2016.

Nikola Mrksic, Ivan Vulic, Diarmuid Ó Séaghdha, Ira Leviant, Roi Reichart, Milica Gasic, Anna Korhonen, and Steve J. Young. Semantic specialization of distributional word vector spaces using monolingual and cross-lingual constraints. *Transactions of the Association for Computational Linguistics*, 5:309–324, 2017.

Anh Nguyen, Jeff Clune, Yoshua Bengio, Alexey Dosovitskiy, and Jason Yosinski. Plug play generative networks: Conditional iterative generation of images in latent space. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3510–3520, 2016.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan T. McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. Universal dependencies v1: A multilingual treebank collection. In *LREC*, 2016.

Chris Olah and Shan Carter. Attention and augmented recurrent neural networks. *Distill*, 2016. doi: 10.23915/distill.00001. URL `http://distill.pub/2016/augmented-rnns`.

Masataka Ono, Makoto Miwa, and Yutaka Sasaki. Word embedding-based antonym detection using thesauri and distributional information. In *HLT-NAACL*, 2015.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 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, 2002.

Frédérik Paradis. Poutyne: A Keras-like framework for PyTorch, 2018. `https://poutyne.org`.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Hao Peng, Bhuwan Dhingra, and Dipanjan Das. Generation with exemplar-based adaptive decoding. 2019.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

Matthew Peters, Sebastian Ruder, and Noah A. Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. *ArXiv*, abs/1903.05987, 2019.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke S. Zettlemoyer. Deep contextualized word representations. *ArXiv*, abs/1802.05365, 2018.

Telmo Pires, Eva Schlinger, and Dan Garrette. How multilingual is multilingual bert? In *ACL*, 2019.

Edoardo Maria Ponti, Ivan Vulic, Goran Glavas, Nikola Mrksic, and Anna Korhonen. Adversarial propagation and zero-shot cross-lingual transfer of word vector specialization. In *EMNLP*, 2018.

Edoardo Maria Ponti, Ivan Vulic, Goran Glavas, Roi Reichart, and Anna Korhonen. Cross-lingual semantic specialization via lexical relation induction. In *EMNLP/IJCNLP*, 2019.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *EACL*, 2017.

Alec Radford. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. Hubs in space: Popular nearest neighbors in high-dimensional data. *JMLR*, 11:2487–2531, December 2010. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1756006.1953015.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *NIPS*, 2017.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP/IJCNLP*, 2019.

Samuel Rönnqvist, Jenna Kanerva, Tapio Salakoski, and Filip Ginter. Is multilingual bert fluent in language generation? *ArXiv*, abs/1910.03806, 2019.

Sascha Rothe and Hinrich Schütze. Autoextend: Extending word embeddings to embeddings for synsets and lexemes. In *ACL*, 2015.

Sebastian Ruder, Ivan Vuli'c, and Anders Sogaard. A survey of cross-lingual word embedding models. 2018.

Erik F. Tjong Kim Sang. Introduction to the conll-2002 shared task: Language-independent named entity recognition. *ArXiv*, cs.CL/0209010, 2002.

Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *ArXiv*, cs.CL/0306050, 2003.

Roy Schwartz, Roi Reichart, and Ari Rappoport. Symmetric pattern based word embeddings for improved word similarity prediction. In *CoNLL*, 2015.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909, 2016.

Louis Shao, Stephan Gouws, Denny Britz, Anna Goldie, Brian Strope, and Ray Kurzweil. Generating long and diverse responses with neural conversation models. *CoRR*, abs/1701.03185, 2017.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. In *ACL*, 2019.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. 2011.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Sho Takase, Jun Suzuki, and Masaaki Nagata. Direct output connection for a high-rank language model. In *EMNLP*, 2018.

Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13:22–30, 2011.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NIPS*, 2015.

Antti Virtanen, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, and Sampo Pyysalo. Multilingual is not enough: Bert for finnish. *ArXiv*, abs/1912.07076, 2019.

Ivan Vulic and Anna Korhonen. On the role of seed lexicons in learning bilingual word embeddings. In *ACL*, 2016.

Ivan Vulić and Nikola Mrkšić. Specialising word vectors for lexical entailment. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1134–1145, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1103. URL https://www.aclweb.org/anthology/N18-1103.

Ivan Vulic, Nikola Mrksic, Roi Reichart, Diarmuid Ó Séaghdha, Steve J. Young, and Anna Korhonen. Morph-fitting: Fine-tuning word vector spaces with simple language-specific rules. In *ACL*, 2017.

Michael E. Wall, Andreas Rechtsteiner, and L. M. Rocha. Singular value decomposition and principal component analysis. 2003.

Alex Wang and Kyunghyun Cho. Bert has a mouth, and it must speak: Bert as a markov random field language model. *CoRR*, abs/1902.04094, 2019.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*, 2018.

Dilin Wang, Chengyue Gong, and Qiang Liu. Improving neural language modeling via adversarial training. *ArXiv*, abs/1906.03805, 2019.

Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Jianshu Ji, Cuihong Cao, Daxin Jiang, and Ming Zhou. K-adapter: Infusing knowledge into pre-trained models with adapters. *ArXiv*, abs/2002.01808, 2020.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina Maria Rojas-Barahona, Pei hao Su, Stefan Ultes, David Vandyke, and Steve J. Young. A network-based end-to-end trainable task-oriented dialogue system. In *EACL*, 2017.

Wikipedia contributors. Byte pair encoding — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Byte_pair_encoding&oldid=931113014, 2019. [Online; accessed 28-December-2019].

Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. Learning neural templates for text generation. In *EMNLP*, 2018.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *ArXiv*, abs/1906.08237, 2019.

Jun Yin, Xin Jiang, Zhengdong Lu, Lifeng Shang, Hang Li, and Xiaoming Li. Neural generative question answering. In *IJCAI*, 2016.

Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. Image captioning with semantic attention. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4651–4659, 2016.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. Defending against neural fake news. *ArXiv*, abs/1905.12616, 2019.