University of Turin
Computer Science Department

Ph.D. Program in Computer Science
Cycle XXXIV

Architectural Augmentations and Sparsity in Deep Language Generation Models

Giovanni BONETTA

Supervisor: Prof. Rossella CANCELLIERE
Ph.D. Program Coordinator: Prof. Viviana PATTI

Academic years of enrollment: 2018/2019
2019/2020
2020/2021
Code of scientific discipline: INF/01

UNIVERSITY OF TURIN

Abstract

School of Science of Nature

Computer Science Department

Doctor of Philosophy

Architectural Augmentations and Sparsity in Deep Language Generation Models

by Giovanni BONETTA

Very often, in Natural Language Processing (NLP), a text input is processed to generate a text output that is related to it, like: a translation from one language to another, a continuation to a given context, an answer to a question, etc. This not only requires the model to broadly understand the input content, but also to gather some specific information from it, like named entities to create accurate outputs. Many deep learning models have great generalization capabilities but luck on accuracy, generating inconsistent or hallucinated outputs. In this work we study this issue, proposing some methods to enhance faithful text output generation and to control hallucinations. Our research mainly focus on two objectives: a copying technique which process the given input information more thoroughly; and an Information Retrieval based method which allows to output more various and interesting utterances. For the former task we propose an algorithm which relies on a language model to read tabular text data and leverages a soft-switch mechanism to decide when it is appropriate to copy from the input, instead of simply generating conditioning on it. We show that the so created model obtains very precise and sensible results in the framework of Data-to-Text (DTT), while keeping the generalization ability. For the latter objective we draw from the observation that a source of information which is usually underused is the training set itself: it is needed to create the model, but then it is no more used at inference time. Therefore we propose to leverage this data over the entire pipeline by merging a Language Model (LM) with a K-Nearest-Neighbors (kNN) algorithm. So doing, we steer the great generation capability of LMs via conditioning on information and facts coming from retrieved text. We test this idea on dialog learning, a notoriously difficult task since not only it requires the models to understand the content in the chat history (which may be very long), but also to choose and generate the best continuations accordingly to the chat tone, arguments and being fact consistent. The resulting system is able to produce more interesting and diverse utterances than the simple LM counterpart and it is less prone to fact errors. In recent years the need for more accurate and reliable solutions in NLP, and maybe within the entire deep learning world, has led to the trend of creating deeper and deeper models in order to make them powerful enough to tackle increasingly complex objectives. Although effective, this idea has the drawback of heading towards overparameterized networks. These big models are very clunky and usually occupy massive amount of memory which, among other things, make it difficult to use them on small hardware, like smartphones or IoT devices, and also asks for complex parallelized environments, very costly to build. We face this problem presenting a technique to reduce the number of parameters used by big deep learning models, allowing them to be stored more easily while still producing comparable results with the overparameterized ones. The idea is to iteratively shrink the models' weights which have less impact on the loss computation. This allows to eventually prune the non-useful weights and keep the network "core". Another benefit of pruning is that the remaining network is sparse and might has better generalization properties. We show how this technique can be used without modifications for tasks very different from each other like dialog learning, machine translation, and even object classification with computer vision models.

Contents

List of Figures

List of Tables

List of Abbreviations

ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
BLEU	**Bi**-**L**ingual **E**valuation **U**nderstudy
CIDEr	**C**onsensus-based **I**mage **D**escription **E**valuation
CNN	**C**onvolutional **N**eural **N**etwork
DTT	**D**ata-**T**o-**T**ext Generation
E2E	**E**nd-to-**E**nd
GPU	**G**raphical **P**rocessing **U**nit
GRU	**G**ated **R**ecurrent **U**nit
HSMM	**H**idden **S**emi **M**arkov **M**odel
IC	**I**mage **C**lassification
kNN	**k**-**N**earest **N**eighbors
LM	**L**anguage **M**odel
LSTM	**L**ong **S**hort-**T**erm **M**emory
METEOR	**M**etric for **E**valuation of **T**ranslation with **E**xplicit **OR**dering
MR	**M**eaning **R**epresentation
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
NLP	**N**atural **L**anguage **P**rocessing
NLG	**N**atural **L**anguage **G**eneration
ReLU	**Re**ctified **L**inear **U**nit
RNN	**R**ecurrent **N**eural **N**etwork
ROUGE	**R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation
SGD	**S**tochastic **G**radient **D**escend
TL	**T**ransfer **L**earning

Chapter 1

Introduction

Natural Language Generation (NLG) is the research subfield which lies in the intersection between artificial intelligence and computational linguistic and has the goal to create computer systems that can produce linguistic outputs in order to meet specified communicative goals. Some examples of NLG tasks are Question Answering (QA), Dialog Generation (DG) or Machine Translation (MT).

In the last few years NLG has experienced a massive growth both in academia and in the number of industrial applications mainly due to two factors: the increasing simplicity to train and deploy robust deep learning models and the growing presence of sources of textual data, which can be found organized in huge datasets.

Anyway the presence of a great amount of textual data sometimes is not enough in order to obtain accurate model performance. Even big deep models trained on such datasets suffer by different flaws. One example is the presence of invented facts in the generated language output, the so called hallucinations; in these cases very often the names of people, locations, phone numbers, addresses and other rare textural entities are wrong or simply non-factually true.

Other examples of unsatisfactory results can be found in the dialog generation field, where it exists the tendency of trained models to complete a given dialog with very general or non-informative next turns. Greetings or simple repetitions of previous utterances are very common and, although grammatically and semantically correct, they lead to dull or inconclusive conversations.

These issues and more lead to a single question: how can we ensure faithful and informative generation using data-driven models?

This thesis tries to answer to this question focusing on using input data in a more sophisticated way, in order to obtain a Language Generation system which produces outputs that do not contain false or trivial informations, while being accurate and coherent.

We deal with this issue in the framework of two main NLG tasks: Data-To-Text generation (DTT) and Dialog Generation.

DTT is an instance of NLG and can be defined as "the problem of generating descriptive text from database records" (Wiseman et al., 2017) which usually can be seen as sets of key-value pairs (Lebret et al., 2016; Novikova et al., 2017). This DTT definition still includes a broad range of applications, including:

Summary of hospital patients conditions: (Banaee et al., 2013; Gatt et al., 2009). Physician's reports or exams' results can be converted into textual summaries for efficient reading.

Current news reports: (Leppänen et al., 2017). These application allows to streamline the creation of news with no human in the loop, or just with minor human interventions.

Weather forecasts: (Goldberg et al., 1994; Ramos-Soto et al., 2015; Reiter et al., 2005; Turner et al., 2007). Data from different weather data sources like sensors and meteorology agencies are orgnized and processed to generate forecasts.

financial reports: (Plachouras et al., 2016). These systems digest financial data, which are usually organized as huge sets of numerical results, and allow to gather information from them by querying these sources in a natural textual way.

Soccer matches chronicle: (D. L. Chen & Mooney, 2008; Theune et al., 2001). Soccer matches are easily described by some key information like the two soccer teams, the match result, the place where the match was played. This data can be converted in a descriptive textual form.

Museum-specific interactive information: (O'Donnell et al., 2001; Stock et al., 2007). Museum exhibits like archaeological finds, paintings, or sculptures, can be organized and presented in interactive environments with textual descriptions of them. (O'Donnell et al., 2001; Stock et al., 2007);.

Traditionally, the DTT problem has been faced via pipeline models, in which each submodule addresses a specific sub-task (detailed in Chapter 2). Symbolic systems were the *de facto* standard, even if they typically require handcrafted rules heavily exploiting domain experts' work, and are very far from being generalizable. The rise of data-driven models, and particularly of Deep Learning-based ones, have naturally brought to a shift in this paradigm with the creation of the so called end-to-end architectures, where the modular pipeline is substituted with a single neural network. Their data-driven design allows the development of general models for DTT, but on the other hand they require a significant amount of data to reach satisfactory performance.

In this context we present in Chapter 2 an approach which effectively takes care of the hallucination problem thanks to the introduction of a copy mechanism which exploits a character-based vocabulary. In many Language Generation tasks, such as Question Answering or Dialog Generation, a deep learning model would be much more effective if it had the ability to directly retrieve named entities or rare words from the input itself, or in other terms, copying them. Moreover, other times the information needed at generation time is not accessible from the input itself but it is present in the training data. If this is the case the tokenization/delexicalization preprocessing of the training data, typical of Language Generation tasks, may lead to the loss of the specific information (i.e. rare words are swapped with the <unk> token) and in turn this leads to a generation miss. This issue can be solved in a simple but rather elegant way, by entirely removing the <unk> token from the vocabulary. We achieve it by relying on a character-based vocabulary which contains upper/lowercased letters, digits and special caracters as used in the ASCII printable standard. So

doing, the <unk> token is no more needed since every single peace of textual data can be expressed as a sequence of characters, and it can be processed in a lossless way.

Dialog Generation is a branch of NLG which aims at creating systems able to model human language and how humans comunicate through dialog. The resulting models have the ability to generate or continue given dialogs in sensible ways. In recent years such systems have been studied and implemented both in academia and industry and are now deployed in many different areas, such as:

Personal assistant systems: (Cortana (Microsoft, 2021), Siri (Apple, 2021)). Their dialogue functions can effectively save users considerable time and effort helping with the completion of various services; from booking a sit at the cinema to get whether forecasts.

Dialog completion systems: (Google Smart Replay (Kannan et al., 2016)). These applications are able to complete some simple dialog or piece of text and are usually deployed in the mobile world as a way to facilitate and speed up the human interaction with mail, social and messaging apps.

Customer service bots: (Five9 Blended Contact Center (Five9, 2021), SmartAssist (Kore.ai, 2021)). They are business platforms able to create conversational engagements with clients and help call center agents completing or suggesting answers to users aiming at faster time to resolution.

Chit-chat chatbots: (I. V. Serban et al., 2017; Y. Zhang et al., 2020). These dialog systems are not goal oriented and try to mimic general dialog generation aiming for interesting conversation and user engagement. They are usually trained on huge chat datasets collected from online resources like forum or social networks logs.

Dialog systems present challenges which are even harder than DTT's. These models must take into account a great amount of context information (dialog history) to understand the tone, arguments and consequently generate coherent continuations without repetitions or misinformed facts. Moreover, generating a dialog continuation requires the model to decide how to steer the dialog itself; when to ask for a question, when to greet, make a joke or suggest interesting facts, having a much more active role than in DTT. Therefore it is not a surprise that the hallucination problem is still very much present in dialog system's outputs and must be taken into account.

As a way to alleviate this issue and in general to build a more reliable, coherent model, we present in chapter 3 a way to combine a LM with a Retrieval System exploiting both the general language related skills of a trained deep learning dialog model and the precision and the informative power of a retrieval platform. We show that this system is able to complete dialogs with the usual dynamism of a well trained chatbot, but also avoiding hallucinations and dull continuations by leveraging data from a given datastore which can very well be, but it is not limited to, the training data itself.

The dimensions of NLP models as the ones just described have constantly grown going from $\sim$ 10s of million to $\sim$ 100s of billions (Brown et al., 2020; Microsoft-Nvidia, 2022) in just a few years. This has led to very slow and memory consuming models which now require sophisticated ways to be trained

and deployed effectively, like the need for parallelization, mixed-precision computations and specific acceleration hardware.

Consequently, as further contribution in this thesis, in chapter 4 we study the possibility to prune deep neural architectures in order to streamline their usage.

Sparsity is the set of techniques that aim at limiting (or decreasing) the dimensionality of a neural network by pruning parameters within the model itself, while maintaining acceptable performance. In practice the sparsity of a weight matrix is the number of zero-valued elements divided by the total number of elements within it; the more the zero parameters the more sparse is the matrix.

Numerous methods for inducing sparsity in the neural networks have been proposed over the past few years: a non-exhaustive review can be found in section 4.1. Among the most popular ones the $L2$ regularization based techniques add a penalty term to the cost functional in order to shrink parameters' values. All parameters dropping below a predefined threshold are then set to zero, thus obtaining sparse architectures. Our method lies in this framework.

A drawback of these methods is that neural weights' norms are all driven close to zero without taking into account of weight relevance in the neural architecture, as discussed in detail in Tartaglione et al. (2022) and Tartaglione et al. (2018).

The pruning technique we deal with in this thesis proposes a new loss functional holding a suited regularization term, and demonstrate that application of the stochastic gradient descent (SGD) algorithm allows to derive a new weights' update rule which selectively decreases the norm of non relevant weights, while performing a classic update for relevant ones. Shrinked weights are then pruned in order to sparsify the neural architecture. This technique is effectively applied to prune two different Transfomer-based models in the context of dialog generation but, because the proposed regularization term can be added to any loss functional regardless of its form, it constitutes a unified framework potentially exploitable for many different applications.

An appendix concludes the thesis showing how this technique can be applied also for pruning convolutional neural networks within the framework of image classification.

Chapter 2

Improving Accuracy in Data-To-Text Generation

Early approaches on DTT relied on static rules hand-crafted by experts which addressed different sub-tasks, so simplifying the global objective of generating natural language from structured data. Reiter and Dale (1997) identify the following six sub-tasks:

Content selection: determining the subset of the input information to include in the generated text.

Text structuring: ordering the information in an accessible way.

Sentence aggregation: splitting the information in sentences, both at the semantic and at the syntactic level.

Lexicalization: deciding the words and phrases that verbalize information.

*Referring Expression Generation (*REG*):* choosing the words and phrases that unambiguously identify domain objects. The essential difference with lexicalization is that REG consists in a discrimination task. This problem is in turn split in the determination of *referential forms* (pronoun, proper name, definite or indefinite description) and referential contents (set of properties that identify the target entity).

Surface realization: generating the final well-formed, syntactically correct sentences. This involves ordering the sentence components, ensuring morphological correctness, producing function words and punctuation. Surface generation typically faces the generation gap and can be interpreted as a mapping between non-isomorphic structures (Ballesteros et al., 2015).

A three-stage pipeline architecture which builds on these tasks' distinction was originally introduced by Reiter (1994) and consists of a *Text Planner*, a *Sentence Planner*, and a *Linguistic Realizer*. This abstract model, outlined by Figure 2.1, has been described as the "de facto standard" (Reiter, 2010; Reiter & Dale, 1997), even if a fair number of systems relax or violate it (Gatt & Krahmer, 2018).

The Text Planner is in charge of content selection and text structuring. It is often referred to as *Macroplanner*, and it determines "what to say". On the other side, the Sentence Planner incorporates sentence aggregation, lexicalization and referring expression generation. In opposite with the Text Planner, it

TABLE 2.1: Most commonly used Data-To-Text Generation datasets and their main features.

Dataset	Domain	Cont. selection	Noisy	Entity
WeatherGov (Liang et al., 2009)	Weather forecast	✓	✗	Single
WikiBio (Lebret et al., 2016)	Biographies	✓	✓	Single
WebNLG (Gardent et al., 2017a)	Various	✗	✗	Single
E2E (Novikova et al., 2017)	Restaurants	✗	✗	Single
RotoWire (Wiseman et al., 2017)	Sportscast	✓	✓	Multiple
SBNation (Wiseman et al., 2017)	Sportscast	✓	✓	Multiple
ToTTo (A. P. Parikh et al., 2020)	Various	✓	✗	Single

is often referred to as *Microplanner*, and it determines "how to say". Finally, the Linguistic Realizer carries out the surface realization task alone.

Unfortunately, splitting the DTT task expose generation systems to the *generation gap* (Meteer, 1991), defined as the presence of mismatches between early and later components, so that antecedent decisions in the pipeline have unexpected, and possibly unfavorable, consequences on the later ones. The problem can be attenuated by merging two or more tasks, such as content selection and text structuring (Duboué & McKeown, 2003), lexicalization and surface realization (Elhadad et al., 1997), or content selection and REG (Engonopoulos & Koller, 2014).

In particular in recent years, neural models blurred the distinction between content selection and surface realization, showing that both can be learned in an end-to-end, data-driven fashion (T. Liu et al., 2019; Mei et al., 2016; Puduppully et al., 2019a). Based on the now-standard encoder-decoder architecture, with attention and copy mechanisms (Bahdanau et al., 2015; Bonetta et al., 2021b; Roberti et al., 2019; See et al., 2017), neural methods for DTT are able to produce fluent text conditioned on structured data in a number of domains (Lebret et al., 2016; Puduppully et al., 2019b; Wiseman et al., 2017), without relying on heavy manual work from field experts.

So, these end-to-end systems solve the DTT generation problem as a whole without the need for splitting it in sub-tasks and therefore avoiding the consequences of the generation gap.

This has been made possible by the advent of larger and more complex datasets (Gardent et al., 2017a; Lebret et al., 2016; Novikova et al., 2017; A. P. Parikh et al., 2020; Wiseman et al., 2017) which satisfy the need for massive data typical of deep learning model training (see Table 2.1 and Table 2.2 for some datasets' features) and this has gone hand in hand with the introduction of larger and more complex benchmarks. In particular, surface-realization abilities have been well studied on hand-crafted datasets such as E2E (Novikova et al., 2017) and WebNLG (Gardent et al., 2017a), while content-selection has been addressed by automatically constructed datasets such as WikiBio (Lebret et al., 2016) or RotoWire (Wiseman et al., 2017).

These large corpora are often constructed from internet sources, which, while easy to access and aggregate, do not consist of perfectly aligned source-target pairs (Dhingra et al., 2019; Perez-Beltrachini & Gardent, 2017). Consequently, model outputs are often subject to over-generation: misaligned fragments from

TABLE 2.2: Size of the Data-To-Text Generation datasets included in Table 2.1

Dataset	No. of instances			Vocabulary size	Avg. target length
	Train	Valid.	Test		
WeatherGov (Liang et al., 2009)	29 528 (total)			345	30.6
WikiBio (Lebret et al., 2016)	582 659	72 831	72 831	∼ 400 000	26.1
WebNLG (Gardent et al., 2017b)	25 298 (total)			8077	22.7
E2E (Novikova et al., 2017)	42 061	4672	4693	∼ 3000	14.3
RotoWire (Wiseman et al., 2017)	3398	727	728	∼ 11 300	337.1
SBNation (Wiseman et al., 2017)	7633	1635	1635	∼ 68 600	805.4
ToTTo (A. P. Parikh et al., 2020)	120 761	7700	7700	136 777	17.4

training instances, namely *divergences*, can induce similarly misaligned outputs during inference, the so-called *hallucinations*.

In this chapter we deal with hallucinations caused by models' inability to correctly copy named entities or rare words from the input itself, which instead are generated from other similar but incorrect entities in training set.

After a brief overview of some significant related works in Section 2.1, sequence to sequence architectures are described in Section 2.2. In Section 2.3 we resume the main ideas on sequence to sequence encoder-decoder architectures with attention.

In Section 2.4 our original contribution to this field is presented (Bonetta et al., 2021b; Roberti et al., 2019), namely a technique which address the rare word issue through a character-based copy mechanism.

Section 2.5 includes the datasets descriptions, some implementation specifications, the experimental framework and the analysis and evaluation of the achieved results.

2.1 Related Works

Popularity and usage of neural models for Data-To-Text generation greatly increased during the last decade. Hereafter we briefly review some important contributions related to the main features and capabilities of our model.

2.1.1 Conditioned Neural Language Models

Wen et al. (2015a) proposed the first neural system specifically designed for this task, which consists in a RNN Language Model (Mikolov et al., 2010), conditioned by a one-hot representation of the input data structure. The model is enriched by a CNN sentence model, which checks the generated sentence for semantic consistency, and by a backward RNN-based reranker. The architecture has been later improved by extending the recurrent LSTM architecture with a gating "sentence planning cell", yielding better results (Wen et al., 2015b).

2.1.2 Standard RNN Encoder-Decoder

An RNN Encoder-Decoder with attention (see Section 2.3) for DTT has been developed by Dusek and Jurcícek (2016), aiming to generate either a deep syntax dependency tree, realized by an external module, or the final sentence, in an end-to-end fashion. Again, a reranking strategy is applied, penalizing the absence of required information and the addition of irrelevant one. As an alternative to reranking, Chisholm et al. (2017) propose an auto-encoding strategy: an RNN Encoder-Decoder with attention is used to generate the natural language description of the input table, and then to get such table from the generated utterance. This constrains output sequences to only express the facts that are present in the data.

2.1.3 Improving Attention Mechanisms

The encoder-aligner-decoder architecture (Mei et al., 2016) is an RNN Encoder-Decoder with a "coarse-to-fine alignment" mechanism. Standard attention weights are re-weighted by the probability of each input token of being selected, computed by a *pre-selector* solely on the basis of the input. This allows a more picky content selection phase. Differently from the previous architectures, the encoder-aligner-decoder takes rid of beam search, reranking and auto-encoding, simply relying on greedy generation. Sha et al. (2018) replace the conventional attention mechanism with a *dispatcher*, that uses a soft switch to choose between the standard content-based attention and a link-based attention, that learns the transition between table fields during decoding, explicitly modeling the generation order of the input fields. A more complex architecture is proposed by Puduppully et al. (2019a), as they interpose a content selection gate and a neural planner between the encoder and the decoder's attention mechanism, that uses the generated plan as the attention keys.

2.1.4 Encoding Structured Data

The main difference between DTT and Machine Translation, i.e. the structured form of the data, has led to work on the encoding side. Lebret et al. (2016) use a novel table encoding and embedding strategy to condition a neural language model, both locally and globally. They also include copy actions, taking into account that input tables often contain output tokens. This encoding strategy has been included in an Encoder-Decoder architecture by T. Liu et al. (2018). Their encoding RNN is a modification of the LSTM cell, in which the cell state is updated using also the field information. Their *dual attention* mechanism uses the product of independent word-based and field-based attention weights to compute the final context vector. Differently, Puduppully et al. (2019b)'s model creates entity representations which are dynamically updated. Their attention mechanism has a hierarchical structure, and it takes into account both the input data and the entity representations. Hierarchical encoders are proposed by T. Liu et al. (2019) and Rebuffel et al. (2021) as well. The former use a word-level and an attribute-level LSTM, and the respective attention weights are combined via an element-wise product. The latter encodes entities from records, and data-structures from entities, taking advantage of Transformer-based architectures. This allows to encode multiple-entity data structures such as the ones included in the RotoWire (Wiseman et al., 2017) dataset.

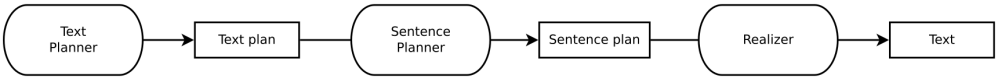


FIGURE 2.1: The classical three-stage pipeline architecture (Reiter, 1994)

2.1.5 Improving Decoding

Compared to the encoder, relatively little work has been focusing on the decoder, indeed generating human-like sentences is not an exclusive property of DTT, unlike encoding data tables. Wiseman et al. (2018) propose a neural reinterpretation of template-based models: their Hidden Semi-Markov Model (HSMM) decoder architecture "learns latent, discrete templates jointly with learning to generate". Such templates are easily interpretable and facilitate the controllability of the generation, even if the quality of the resulting sentences is quite far from state-of-the-art models.

2.2 Sequence-to-Sequence Architectures

Sequence-to-sequence (Seq2Seq) frameworks (Aharoni et al., 2016; Cho et al., 2014; Sutskever et al., 2014) have proved to be very effective in NLG tasks (Karpathy & Li, 2015; Mei et al., 2016; Wen et al., 2015a), as in Machine Translation (Bahdanau et al., 2015; Cho et al., 2014; Sennrich et al., 2016; Sutskever et al., 2014) and in Language Modeling (Al-Rfou et al., 2019).

One of the most effective declination of the sequence to sequence architecture is the Encoder-Decoder with Recurrent Neural Networks (Cho et al., 2014; Sutskever et al., 2014) which consists of two separate RNNs, the *encoder* and the *decoder*, which play different roles; the encoder reads the input and the decoder generates the output.

The Encoder-Decoder models (and Seq2Seq in general) assume the data they are fed with to be a sequence, but one of the main difference between Data-To-Text and other Text-To-Text tasks, such as the ones just cited, is the non-linear structure of the input. Data tables can have either a key-value structure, interchangeably called slot-value structure, (Lebret et al., 2016; Novikova et al., 2017), or a more complex table form (Wiseman et al., 2017), so they need a preprocessing step called *linearization*, which makes them compatible with neural Seq2Seq systems. Linearization of the input involves (i) creating embedding vectors that encode it in a convenient way, and (ii) determining an arbitrary order for inherently unordered data.

Key-value pairs can be treated as independent subsequent embedded tokens, delegating to the network the task of distinguishing tokens belonging to data keys from those belonging to the corresponding values (Dusek & Jurcícek, 2016). This sub-task can be made more trivial to the network by using some special tokens as delimiters for key and/or values. This is also the approach used in this thesis since we use the "[" and "]" characters as value delimiters. A different approach consists in concatenating the key embedding and each value token's one (Sha et al., 2018), possibly adding a linear projection and a non-linear activation, such as the hyperbolic tangent (Wiseman et al., 2017; Yang et al., 2017). The representation of the key relative to a given token can

TABLE 2.3: Special tokens included in the vocabulary $\mathcal{V}$, and their respective roles.

Token	Description
<s>	Start of sequence
</s>	End of sequence
<unk>	Unknown, out-of-vocabulary word
<pad>	Padding (used in mini-batching)

be enriched by such token's position, counted from both the start and the end of the sequence (Lebret et al., 2016; T. Liu et al., 2018).

Different ordering of key-value pairs during training impacts the resulting generation systems' performance (Kedzie & McKeown, 2020). In particular, when the order of the pairs matches the output sentence's realization order, models tend to be more controllable.

Once the input data are correctly formatted and organized for the Encoder-Decoder model the training/inference process can begin. Neural Seq2Seq architectures take a sequence $\{x_1, \ldots, x_{T_x}\}$ as input, and output another sequence $\{y_1, \ldots, y_{T_y}\}$, where the sequences' lengths are T_x and T_y, respectively. Input sub-sequences ranging from 1 to j are referred to as $x_{1:j}$ so that $x = x_{1:T_x}$. Similarly, output sub-sequences ranging from 1 to t are referred to as $y_{1:t}$, and $y = y_{1:T_y}$.

More specifically, in Data-To-Text Generation, inputs are variable-sized sets of key-value pairs $\langle k; w_{1:T_k} \rangle$. Input and output sequences share the same *vocabulary* $\mathcal{V}$, defined as the set of all possible $V = |\mathcal{V}|$ tokens, including the special ones shown in Table 2.3. Both input and output sequences are linearized and transformed in lists of embedded tokens prior to be fed into the neural model.

Hereafter we indicate with W and b the model's weight matrices and vectors respectively, whose values are learned via back-propagation. Hidden states size (or *model sizes*) is generically referred to as $emb \in \mathbb{N}^+$.

The encoder RNN is in charge of reading the input sequence one time step j at a time, updating its hidden state vector $h_j \in \mathbb{R}^{emb}$ as described in eq. 2.1.

$$h_j = \text{RNN}_{\text{enc}}(x_j, h_{j-1}), \quad j = 1, \ldots, T_x. \tag{2.1}$$

The decoder RNN updates its hidden state d_t conditioned by the previous generated token and d_{t-1}. Moreover, the encoder's final hidden state is used to bootstrap the decoder (i.e. $d_0 = h_{T_x}$):

$$d_t = \text{RNN}_{\text{dec}}(y_{t-1}, d_{t-1}), \quad t = 1, \ldots, T_y. \tag{2.2}$$

At each time step t, the corresponding decoder's hidden state d_t is projected to a vocabulary-sized vector o_t, which is in turn converted to a categorical probability distribution:

$$o_t = W \cdot d_t + b \tag{2.3}$$

$$P(y_t | y_{1:t-1}, x) = \text{softmax}(o_t), \tag{2.4}$$

where $W \in \mathbb{R}^{emb \times V}$. The output token y_t is generated according to the probability distribution $P(y_t|y_{1:t-1}, x)$.

Note that the Encoder inputs x_j and the Decoder inputs y_{t-1} are transformed into embedding vectors, which are stored in a lookup table and they are learned during training via a linear transformation of a randomly initialized matrix $E \in \mathbb{R}^{V \times emb}$, where each row corresponds to a token.

A typical RNN Encoder-Decoder shows the following features:

- the RNN variants used for both the encoder and the decoder are Long Short-Term Memory (Gers et al., 2000; Hochreiter & Schmidhuber, 1997) or, less frequently, Gated Recurrent Units (Cho et al., 2014). Those architectures reduce the exploding or vanishing gradient problems (Bengio et al., 1993; Bengio et al., 1994) and better deal with long-term dependencies inside sequences.
- the encoder is bidirectional (Schuster & Paliwal, 1997), as the whole input sequence is typically available and information from both left and right tokens can be informative.
- the decoder uses input feeding, i.e. "attentional vectors are fed as inputs to the next time steps to inform the model about past alignment decisions". This aspect is better presented in Sec. 2.3.
- the whole architecture is trained end-to-end, using Back-Propagation Through Time (Rumelhart et al., 1986; Williams & Zipser, 1989) and Teacher Forcing (Williams & Zipser, 1989).
- the inference phase can be carried out using different decoding techniques which usually differs by the sampling process used to generate the next output token. The most used ones are greedy, beam search (Dept., 2015), top-k (Fan et al., 2018) and nucleus sempling (Holtzman et al., 2020).

2.3 Encoder-Decoder Architectures with Attention

The Seq2Seq architecture just described in 2.2, although being very effective, shows a major drawback: it can not scale easily to very long sequences. The sequential nature of RNNs leads to a difficult gradient flow during the learning phase and it gets worsen for long inputs/outputs, with issues like the vanishing or exploding gradient (Pascanu et al., 2013). Moreover, since the autoregressive decoder is conditioned on the last encoder hidden state h_{T_x} , the amount of information which flows from the encoder to the decoder is limited by the dimension of h_{T_x}.

The Encoder-Decoder architecture with Attention module (or mechanism) (Bahdanau et al., 2015) is a renewed way to solve these issue. This neural net component aims to emulate the human unconscious attention mechanism. For example, during a first vision of an image, we focus our attention on a precise region of the image itself; once we have inferred what is present, we move on to another region, but with expectations derived from the first inference. Similarly, when reading a sentence, attention falls on verbs, which allow us to decode the context quickly and predict (in part) what will come next. We can

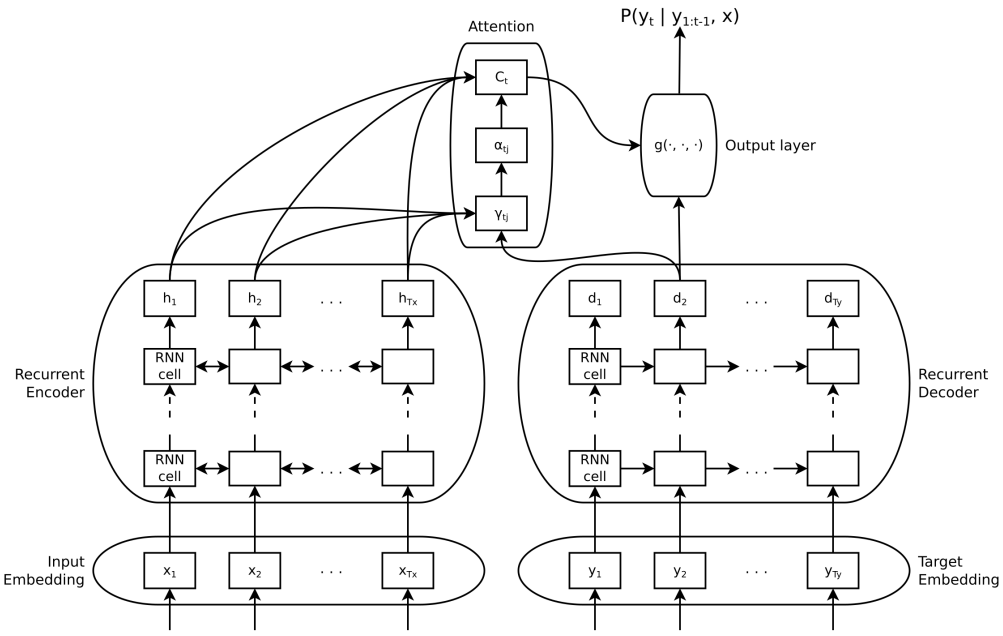


FIGURE 2.2: The Encoder-Decoder with attention (Bahdanau et al., 2015; T. Luong et al., 2015). Left side: the encoder is implemented as a bi-directional RNN and outputs one annotation h_j for each input token x_j. Right side: the decoder, which produces one state d_t for each time step. At the top the attention mechanism is shown.

see Attention as a vector of importance weights, which puts in first place the portions of data relevant to the task we want to learn.

The Encoder-Decoder architecture with Attention, which we may simply refer to as EDA, is represented in Figure 2.2. The main components of the attention mechanism are:

(i) the alignment model e_{tj}

$$e_{tj} \equiv att(d_{t-1}, h_j) = b^T \cdot \tanh(W_a \cdot [d_{t-1}; h_j]), 1 \leq j \leq T_x, \;\; 1 \leq t \leq T_y \quad (2.5)$$

which is parameterized as a feedforward neural network and scores how well input in position j-th and output observed in the t-th time instant match. T_x and T_y are the length of the input and output sequences, respectively. Each vector h_j corresponds to the concatenation of the hidden states produced by the backward and forward RNNs of the bidirectional encoder.

(ii) the attention probability distribution α_{ij}

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T_x} \exp(e_{tk})} \equiv [softmax(e_t)]_j, \quad 1 \leq j \leq T_x, \;\; 1 \leq t \leq T_y \quad (2.6)$$

(e_t is the vector whose j-th element is e_{tj})
(iii) the context vector C_t

$$C_t = \sum_{j=1}^{T_x} \alpha_{tj} h_j, \quad 1 \leq t \leq T_y, \quad (2.7)$$

weighted sum of the encoder annotations h_j.

According to Bahdanau et al., 2015, the context vector C_t is the key element for evaluating the conditional probability $P(y_t|y_1, \ldots, y_{t-1}, x)$ to output a target token y_t, given the previously outputted tokens $y_1, \ldots, y_{t-1}$ and the input x. They in fact express this probability as:

$$P(y_t|y_1, \ldots, y_{t-1}, x) = g(y_{t-1}, d_t, C_t), \tag{2.8}$$

where g is a non-linear, potentially multi-layered, function. So doing, the explicit information about $y_1, \ldots, y_{t-1}$ and x is replaced with the knowledge of the context C_t and the decoder state s_t.

2.4 Learning to Copy

Encoder-Decoder models uses data represented in a word-by-word scheme both in input and output sequences; anyways, such schemes can't be effective without a special, non-neural *delexicalization* phase that handles unknown words, such as proper names or foreign words (Wen et al., 2015b). The delexicalization step has the benefit of reducing the dictionary size and, consequently, the data sparsity, but it is affected by various shortcomings. In particular, according to Goyal et al. (2016) - it needs some reliable mechanism for entity identification, i.e. the recognition of named entities inside text; - it requires a subsequent *re-lexicalization* phase, where the original named entities take back placeholders' place; - it cannot account for lexical or morphological variations due to the specific entity, such as gender and number agreements, that can't be achieved without a clear context awareness.

Recently, some strategies have been proposed to solve these issues: Gu et al. (2016) and See et al. (2017) face this problem using a special neural *copying* mechanism that is quite effective in alleviating the out-of-vocabulary words problem, while T. Luong et al. (2015) tries to extend neural networks with a post-processing phase that copies words as indicated by the model's output sequence. Some character-level solutions of this issue have been presented as well, either as a fallback for rare words (M. Luong & Manning, 2016), or as subword units (Sennrich et al., 2016).

A significantly different approach consists in employing characters instead of words, for input slot-value pairs tokenization as well as for the generation of the final utterances, as done for instance in Agarwal and Dymetman (2017) and Al-Rfou et al. (2019).

The model we dealt with lies in this framework (Bonetta et al., 2021b; Roberti et al., 2019): it is a character-level Encoder-Decoder model with attention mechanism and copying capabilities that resulted in a completely neural end-to-end architecture. In contrast to traditional word-based ones, it does not require delexicalization, tokenization nor lowercasing; besides, according to our experiments it never hallucinates words, nor duplicates them. As we will see, such an approach achieves rather interesting performance results and produces a vocabulary-free model that is inherently more general, as it does not depend on a specific domain's set of terms, but rather on a general alphabet. Because of this, it opens up the possibility, not viable when using words, to adapt already trained networks to deal with different datasets.

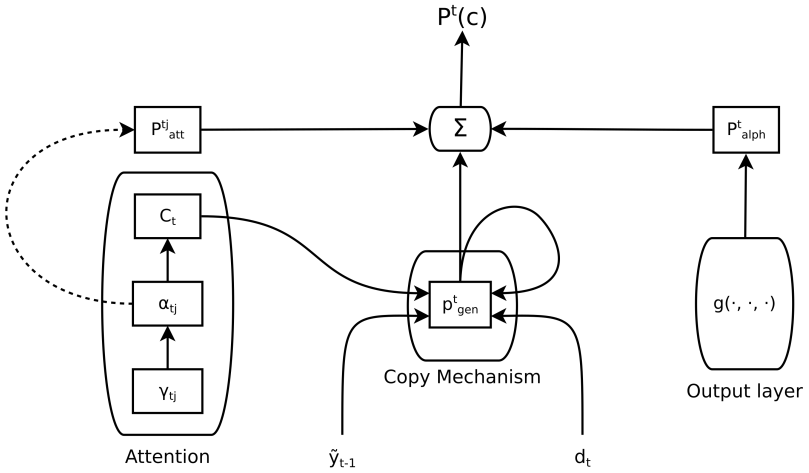


FIGURE 2.3: Our proposed copy mechanism: the final probability distribution is the sum of P^t_{alph} and P^{tj}_{att}, weighted by p^t_{gen}.

More specifically, our model shows two important features, with respect to the architecture proposed by Bahdanau et al. (2015): (i) a character-wise copy mechanism, consisting in a soft switch between generation and copy mode, that disengages the model to learn rare and unhelpful self-correspondences, and (ii) a peculiar training procedure, which improves the internal representation capabilities, enhancing recall; it consists in the exchange of encoder and decoder RNNs, – GRUs (Cho et al., 2014) in our specific case – , depending on whether the input is a tabular Meaning Representation (MR [1]) or a natural language sentence.

Our character-based copy mechanism is inspired by the Pointer-Generator Network (See et al., 2017), a word-based model that hybridizes the Encoder-Decoder traditional model and a Pointer Network (Vinyals & Le, 2015). Basing on these ideas, in our model we identify two probability distributions that, differently from what done by See et al. (2017) and Wiseman et al. (2017), *act now on characters* rather than on words: the alphabet distribution P_{alph} and the attention distribution P_{att}.

The former is the network's generative probability of sampling a given character at time t, recalled in eq. (2.8):

$$P^i_{alph} = softmax(W[d_t; C_t] + b), \tag{2.9}$$

where W and b are trainable parameters.

The latter is the distribution reminded in eq. (2.6), created by the attention mechanism over the input tokens, i.e. in our case, over input characters:

$$P^{tj}_{att} \equiv \alpha_{tj} \tag{2.10}$$

In our method this distribution is used for directly copying characters from the input to the output, pointing their input positions, while in (Bahdanau et al., 2015) P_{att} is used only internally to weigh the input annotations and create the context vector C_t.

[1] MR is another way to indicate a set of attributes in the form of key-value pairs.

The final probability of outputting a specific character c is obtained combining P_{alph} and P_{att} through the quantity p_{gen}, defined later, which acts as a soft switch between generating c or copying it (see fig. 2.3):

$$P^t(c) = p^t_{gen} \cdot P^t_{alph}[c] + (1 - p^t_{gen}) \sum_{j|x_t=c} P^{tj}_{att}(c), \tag{2.11}$$

where $P^t_{alph}[c]$ is the component of P^t_{alph} corresponding to that character c.

The backpropagation training algorithm, therefore, brings p_{gen} close to 1 when it is necessary to generate the output as in a standard Encoder-Decoder with Attention ($P^t(c) \simeq P^t_{alph}[c]$); conversely, p_{gen} will be close to 0 (i.e. $P^t(c) \simeq \sum_{j|x_t=c} P^j_{att}(c)$) when a copying step is needed.

The model we propose therefore learns when to sample from P_{alph} for selecting the character to be generated, and when to sample from P_{att} for selecting the character that has to be copied directly from the input.

This copy mechanism is fundamental to output all the unknown words present in the input, i.e. words which never occur in the training set. In fact, generating characters in the right order to reproduce unknown words is a sub-task not "solvable" by a naive Seq2Seq model, which learns to output only known words.

The generation probability $p_{gen} \in [0, 1]$ is computed as follows:

$$p^t_{gen} = \sigma(W_y \cdot \tilde{y}_{t-1} + W_d \cdot d_t + W_p \cdot p^{t-1}_{gen} + W_c \cdot C_t) \tag{2.12}$$

where σ is the sigmoid function, $\tilde{y}_{t-1}$ is the last output character's embedding, d_t is the current decoder's cell state and C_t is the current context vector. W_y, W_d, W_c and W_p are the parameters whose training allows p_{gen} to have the convenient value.

We highlight that in our formulation p^{t-1}_{gen}, i.e. the value of p_{gen} at time $t-1$, contributes to the determination of p^t_{gen}. In fact, in a character-based model it is desirable that this probability remains unchanged for a fair number of time steps, and knowing its last value helps this behavior. This never happens in word-based models such as See et al. (2017), in which copying for a single time step is usually enough.

We also help the model to learn when it is necessary to start a copying phase, using the following formulation of $P(c)$ (Bonetta et al., 2021b):

$$P^t(c) = p^t_{gen} \cdot P^t_{alph}(c) + (1 - p^t_{gen}) \sum_{j|x_j=c} P^{t,j-1}_{att}(c) \tag{2.13}$$

Sometimes, our model has difficulty in focusing on the first letter it has to copy. This may be caused by the variety of characters it could be attending on; instead, it seems easier to learn to focus on the most largely seen characters, as for instance " " and "[". Since these special characters are very often the prefix of the words we need to copy, when this focus is achieved, we would like the attention distribution to be translated one step to the right, over the first letter that must be copied. Therefore, the final probability of outputting

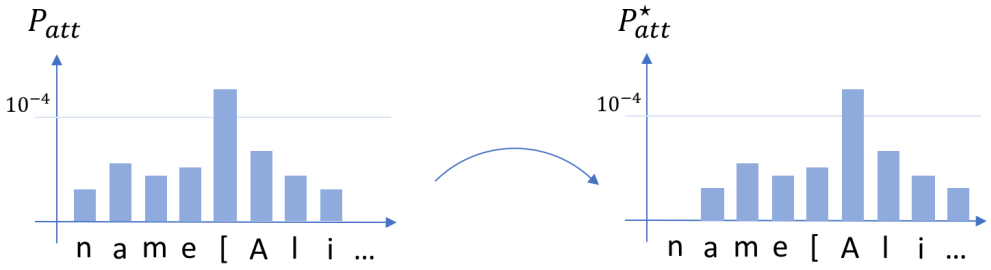


FIGURE 2.4: An example of shifting the attention distribution. On the left the P_{att} distribution is picked on the "[" token, used as delimiter. Shifting the distribution one step to the right allows to pick the distribution on the entity we want to copy.

a specific character c, introduced in eq. (2.11), is modified to $P^{t,j-1}_{att}$, i.e. the attention distribution shifted one step to the right and normalized. Figure 2.4 shows the convenience of this approach.

Notice that $P^{t,j-1}_{att}$ is the only shifted probability, while P^{t}_{alph} remains unchanged. Therefore, if the network is generating the next token (i.e. $p^{t}_{gen} \simeq 1$), the shift trick does not involve $P^{t}(c)$ and the network samples the next character from P^{t}_{alph}, as usual. This means that the shift operation is not degrading the generation ability of the model, whilst improving the copying one.

2.4.1 Switching GRUs

In order to further improve performance, we enrich our model's training pipeline with an additional phase which forces an appropriate language representation inside the recurrent components of the model. In order to achieve this goal, the encoder and the decoder *do not own a fixed GRU*, differently from what happens in classical end-to-end approaches. The recurrent module is passed each time as a parameter, depending on which one of the two training phases is actually performed. So doing, both RNNs learn to encode and decode, becoming more robust

In the first phase, similar to the usual one, the GRU assigned to the encoder, named RNNx, deals with a tabular representation x as input, the GRU assigned to the decoder, named RNNy, has to cope with natural language, and the model generates an output utterance $\tilde{y} = F(x)$. Conversely, in the second phase, GRUs are switched and we use as input the just obtained natural language utterance $\tilde{y}$ to generate a new table $\tilde{x} = G(\tilde{y}) = G(F(x))$. The same model can therefore build both F and G, thanks to the switch of GRUs, as shown in Figure 2.5.

In other words, the learning iteration is performed as follows.

- A dataset example (x, y) is given. x is a tabular meaning representation and y is the corresponding reference sentence.
- We generate an output utterance $\tilde{y} = F(x)$ where RNNx is within the encoder and RNNy within the decoder.
- We perform an optimization step on the model's parameters, aiming at minimizing $L_{forward} = loss(\tilde{y}, y)$.

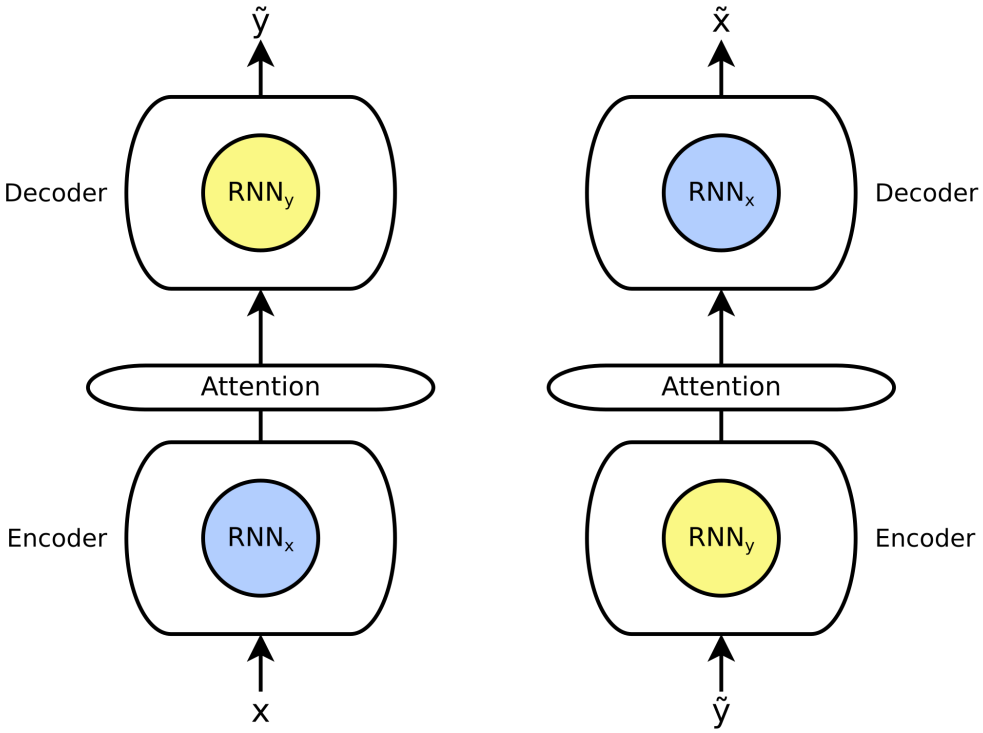


FIGURE 2.5: The switching GRUs mechanism. On the left side the RNNx is used as Encoder and RNNy for the Decoder while on the right side the two GRUs are switched.

- We reconstruct the meaning representation $\tilde{x}$ back from the previously generated output: $\tilde{x} = G(\tilde{y}) = G(F(x))$. So in this case RNNx is used within the decoder and RNNy within the encoder.
- We perform a further optimization step on the model's parameters, this time aiming at minimizing $L_{backward} = loss(\tilde{x}, x)$

The higher training time, direct consequence of the just described technique, is a convenient investment, as it brings an appreciable improvement of the model's performance (see Section 2.5.4).

2.5 Experiments

2.5.1 Datasets

We tested our model on four datasets, whose main descriptive statistics are given in Table 2.4: among them, the most known and frequently used in literature is the E2E dataset (Novikova et al., 2017), used as benchmark for the E2E Challenge [2] organized by the Heriot-Watt University in 2017. It is a crowd-sourced collection of roughly 50,000 instances, in which every input is a list of slot-value pairs[3] and every expected output is the corresponding natural language description. The dataset has been partitioned by the challenge organizers in predefined training, validation and test sets, conceived for training

[2] challenge website: http://www.macs.hw.ac.uk/InteractionLab/E2E/

[3] the *slot-value* term is used with the same meaning of the more known key-value term.

data-driven, end-to-end Natural Language Generation models in the restaurant domain. A dataset sample is shown in Table 2.5.

TABLE 2.4: Descriptive statistics. Left side: sizes of training, validation and test sets. Right side: average number of characters, respectively for MR and natural language sentences.

Dataset	Number of instances			Avg. number of characters	
	training	validation	test	MRs	NL sentences
E2E	42061	4672	4693	112.11	115.07
E2E+	42061	4672	4693	112.91	115.65
Hotel	2210	275	275	52.74	61.31
Restaurant	2874	358	358	53.89	63.22

However, during our experiments, we noticed that the values contained in the E2E dataset are a little naive in terms of variability. In other words, a slot like *name*, that could virtually contain a very broad range of different values, is filled alternating between 19 fixed possibilities. Moreover, values are partitioned among training, validation and test set, in such a way that test set always contains values that are also present in the training set.

In order to better highlight improvements in copying/recalling abilities, and as a further original contribution in our work, we created a modified version of the E2E dataset, called E2E+, as follows: we selected the slots that represent more copy-susceptible attributes, i.e. *name*, *near* and *food*, and conveniently replaced their values, in both meaning representations and reference sentences. New values for *food* are picked from Wikipedia's list of adjectival forms of countries and nations[4], while both *name* and *near* are filled with New York restaurants' names contained in the Entree dataset presented in Burke et al. (1997). It is worth noting that none of the values of *name* are found in *near*; likewise, values that belong to the training set are not found in the validation set nor in the test one, and vice versa. This value partitioning shall ensure the

[4]https://en.wikipedia.org/wiki/List_of_adjectival_and_demonymic_forms_for_countries_and_nations, consulted on August 30, 2018

TABLE 2.5: An E2E data instance. The Meaning Representation appears in the dataset once for each reference sentence.

Meaning Representation	References
name[The Wrestlers], eatType[coffe shop], food[Indian] priceRange[less than L20] area[city centre] familyFriendly[yes] near[Raja Indian Cuisine]	Indian food meets coffee shop at The Wrestlers located in the city centre near Raja Indian Cuisine. This shop is family friendly and priced at less than 20 pounds. Near Raja Indian Cuisine, The Wrestlers provides the atmosphere of a coffee shop with Indian food. At less than 20 pounds, it provides a family friendly setting for its customers right in the city centre. The Wrestlers is a coffee shop providing Indian food in the less than L20 price range. It is located in the city centre. It is near Raja Indian Cuisine.

absence of generation bias in the copy mechanism, stimulating the models to copy attribute values, regardless of their presence in the training set. The *MR* and *1st reference* fields in Table 2.9 are instances of this new dataset.

Finally, we tested our model also on other two datasets, Hotel and Restaurant, frequently used in literature (for instance in Wen et al., 2015a and Goyal et al., 2016). They are built on a 12 attributes ontology: some attributes are common to both domains, while others are domain specific. Every MR is a list of key-value pairs enclosed in a dialogue act type, such as *inform*, used to present information about restaurants, *confirm*, to check that a slot value has been recognized correctly, and *reject*, to advise that the user's constraints cannot be met. For the sake of compatibility, we filtered out from Hotel and Restaurant all inputs whose dialogue act type was not *inform*, and removed the dialogue act type. Besides, we changed the format of the key-value pairs to E2E-like ones.

2.5.2 Metrics

We evaluated the models' performance on test sets' output utterances using the Evaluation metrics script[5] provided by the E2E Challenge organizers. It rates quality according to five different metrics:

- BLEU (Papineni et al., 2002), a length-penalized precision score over n-grams, $n \in [\![1, 4]\!]$, optionally improved with a smoothing technique (B. Chen & Cherry, 2014).
- NIST (Doddington, 2002), a variant of BLEU which gives more credit to rare n-gram and less credit to common ones.
- METEOR (Banerjee & Lavie, 2005), that tries to overcome the fact that BLEU does not take recall into account, and it only allows exact n-gram matching. Hence, METEOR uses the F-measure and a relaxed matching criterion.
- ROUGE_L (C.-Y. Lin, 2004), based on a variation of the F-measure where the precision and recall are computed using the length of the longest common subsequence between hypothesis and reference.
- CIDER (Vedantam et al., 2015), that weighs each hypothesis's n-gram based on its frequency in the reference set and in the entire corpus. The underlying idea is that frequent dataset's n-grams are less likely to be informative/relevant.

2.5.3 Implementation Details

We developed our model using the PyTorch framework[6], release 0.4.1[7].

Tables are encoded simply converting all characters to ASCII and feeding every corresponding index to the encoder, sequentially. The resulting model's vocabulary is common for every possible training set, so making possible the use of transfer learning procedures, where appropriate.

[5] https://github.com/tuetschek/E2E-metrics

[6] Code and datasets are publicly available at https://github.com/marco-roberti/char-data-to-text-gen

[7] https://pytorch.org/

TABLE 2.6: Model hyperparameters and training settings.

Hyperparameter	Value
Embedding size	16
GRU hidden size	256
N. of recurrent layers	4
Attention size	128
Learning rate	10^{-4}
$\beta_1; \beta_2$ for Adam (Kingma & Ba, 2015)	0.9; 0.999
Max gradient norm (Pascanu et al., 2013)	1
Batch size	16
Max no. of epochs	30

The training has been carried out using one Nvidia Titan RTX GPU, as described in Sec 2.4. The two GRUs have the same dimensions, in terms of input size, hidden size, number of layers and presence of a bias term. Moreover, they both have to be bidirectional, even if the decoder ignores the backward part of its current GRU. A typical run lasts for about 10 hours. The final hyperparameters for our model are presented in Table 2.6.

We minimize the negative log-likelihood loss using teacher forcing (Williams & Zipser, 1989) and Adam (Kingma & Ba, 2015); the latter being an optimizer that computes individual adaptive learning rates. As a consequence of the length of the input sequences, a character-based model is often subject to the exploding gradient problem, that we solved via the well-known technique of gradient norm clipping (Pascanu et al., 2013).

2.5.4 Results and Discussion

In order to show that our model represents an effective and relevant improvement, we carry out two different experimentations: an ablation study and a comparison with two well-known models. The first model is the Encoder-Decoder architecture with Attention mechanism by Bahdanau et al., 2015 (hereafter "EDA"), used character-by-character. The second one is TGen (Dusek & Jurcícek, 2016), a word-based model, still derived from EDA, but integrating a beam search mechanism and a reranker over the top k outputs, in order to disadvantage utterances that do not verbalize all the information contained in the MR. We chose it because it has been adopted as baseline in the E2E Challenge and we used its official code implementation. Hyperparameter tuning is done through 10-fold cross-validation, using the BLEU metric Papineni et al., 2002 for evaluating each model. The training stopping criterion was based on the absence of models' performance improvements.

Our first experimentation, the **ablation study**, refers to the E2E dataset because of its wide diffusion, and is shown in Table 2.7; "EDA_CS" identifies our model, and 'C' and 'S' stand for "Copy" and "Switch", the two major improvements presented in this work. It is evident that the partially-improved networks are able to provide independent benefits to the performance. Those components cooperate positively, as EDA_CS further enhances those results. Furthermore, the obtained BLEU metric value on the E2E test set would allow

TABLE 2.7: Ablation study on the E2E dataset. Best values for each metric are highlighted (the higher the better)

EDA	BLEU	0.4999	EDA_S	BLEU	0.6538
	NIST	7.1146		NIST	8.4601
	METEOR	0.3369		METEOR	0.4337
	ROUGE_L	0.5634		ROUGE_L	0.6646
	CIDER	1.3176		CIDER	1.9944
EDA_C	BLEU	0.6255	EDA_CS	BLEU	**0.6705**
	NIST	7.7934		NIST	**8.5150**
	METEOR	0.4401		METEOR	**0.4449**
	ROUGE_L	0.6582		ROUGE_L	**0.6894**
	CIDER	1.7286		CIDER	**2.2355**

TABLE 2.8: Performance comparison. Note the absence of transfer learning on dataset E2E+ because in this case the training and fine-tuning datasets are the same. Best values for each metric are highlighted (the higher the better)

		E2E+	E2E	Hotel	Restaurant
EDA	BLEU	0.3773	0.4999	0.4316	0.3599
	NIST	5.7835	7.1146	5.9708	5.5104
	METEOR	0.2672	0.3369	0.3552	0.3367
	ROUGE_L	0.4638	0.5634	0.6609	0.5892
	CIDER	0.2689	1.3176	3.9213	3.3792
TGen	BLEU	**0.6292**	0.6593	0.5059	0.4074
	NIST	**9.4070**	**8.6094**	7.0913	6.4304
	METEOR	0.4367	0.4483	0.4246	0.3760
	ROUGE_L	**0.6724**	0.6850	0.7277	0.6395
	CIDER	2.8004	2.2338	5.0404	4.1650
EDA_CS	BLEU	0.6197	**0.6705**	0.5515	0.4925
	NIST	9.2103	8.5150	**7.4447**	6.9813
	METEOR	**0.4428**	0.4449	0.4379	0.4191
	ROUGE_L	0.6610	**0.6894**	0.7499	0.7002
	CIDER	**2.8118**	**2.2355**	5.1376	4.7821
EDA_CS^{TL}	BLEU	-	0.6580	**0.5769**	**0.5099**
	NIST	-	8.5615	7.4286	**7.3359**
	METEOR	-	**0.4516**	**0.4439**	**0.4340**
	ROUGE_L	-	0.6740	**0.7616**	**0.7131**
	CIDER	-	2.1803	**5.3456**	**4.9915**

our model to be ranked fourth in the E2E NLG Challenge, while its baseline TGen was ranked tenth.

Our second experimentation, the **comparison study**, is shown in Table 2.8. The character-based design of EDA_CS led us to explore in this context also a possible behavior as a transfer learning capable model (for more information on transfer learning see Goodfellow et al. (2016) and Ruder et al. (2019)). In order to test this hypothesis, we used the weights learned during training on

the E2E+ dataset as the starting point for a fine-tuning phase on all the other datasets. We chose E2E+ because it reduces the generation bias, as discussed in Section 2.5.1. We named this approach EDA_CS^{TL}.

A first interesting result is that our model EDA_CS always obtains higher metric values with respect to TGen on the Hotel and Restaurant datasets, and three out of five higher metrics values on the E2E dataset. However, in the case of E2E+, TGen achieves three out of five higher metrics values. These results suggest that EDA_CS and TGen are comparable, at least from the point of view of automatic metrics' evaluation.

A more surprising result is that the approach EDA_CS^{TL} allows to obtain better performance with respect to training EDA_CS in the standard way on the Hotel and Restaurant datasets (for the majority of metrics). On the E2E dataset, EDA_CS^{TL} outperforms EDA_CS only in one case (i.e. METEOR metric) but shows a BLEU increment of at least 14% with respect to TGen's score when compared to both Hotel and Restaurant datasets.

Finally, the baseline model, EDA, is largely outperformed by all other examined methods.

Therefore, our model exploits its transfer learning capabilities effectively, showing very good performance in a context like Data-To-Text generation in which the portability of features learned from different datasets, in the extent of our knowledge, has not yet been explored.

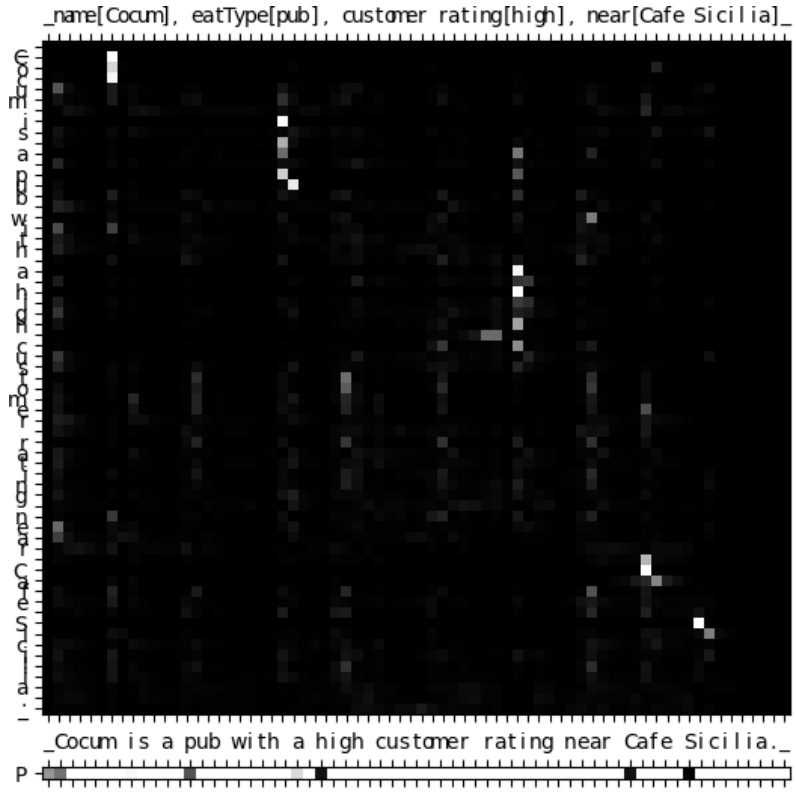
We highlight that EDA_CS's good results are achieved even if it consists in a fully end-to-end model which does not benefit from the delexicalization-relexicalization procedure, differently from TGen. Most importantly, the latter represents a word-based system: as such, it is bound to a specific, limited vocabulary, in contrast to the general-purpose character one used in our work.

Table 2.9 reports the output of the analyzed models for a couple of MR, taken from the E2E+ test set. The EDA's inability to copy is clear, as it tends, in its output, to substitute those values of *name*, *food* and *near* that do not appear in the training set with known ones, guided by the first few characters of the input slot's content. Besides, it shows serious coverage issues, frequently 'forgetting' to report information, and/or repeating the same ones.

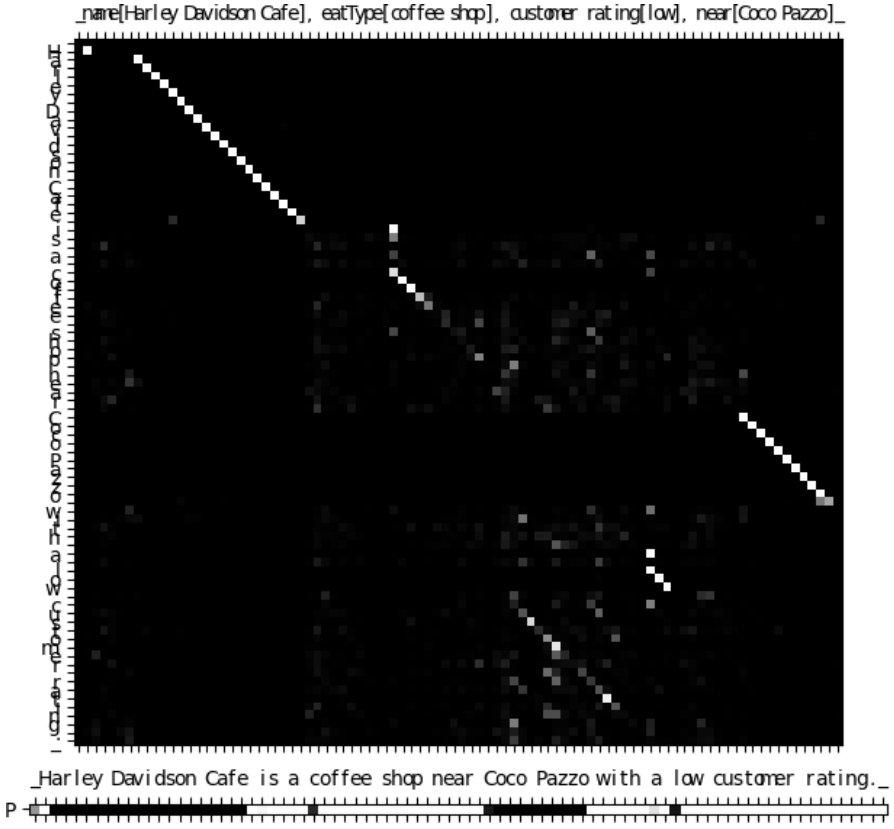
These troubles are not present in EDA_CS output utterances: the model nearly always renders all of the input slots, still without duplicating any of them. This goal is achieved even in absence of explicit coverage techniques thanks to our peculiar training procedure with switching Grus, detailed in Section 2.4, that for each input sample minimizes also the loss on the reconstructed tabular input. It is worth noting that the performance of TGen and EDA_CS are overall comparable, especially when they deal with names or other expressions not present in training.

The joint analysis of the matrix of the attention distribution P_{att}^{tj} and the vector p_{gen} allows a deeper understanding of how our model works.

In Figure 2.6 every row shows the attention probability distribution "seen" when an output character is produced at the t-th time instant (i.e. the vector $P_{att}^{tj}, 1 \leq j \leq T_x$), while every column shows values of the attention distribution corresponding to a specific input position j (i.e. the vector $P_{att}^{tj}, 1 \leq t \leq T_y$).



(A) *On an E2E instance.*



(B) *On an E2E+ instance.*

FIGURE 2.6: Attention distribution (white means more attention) and p_{gen} (white: generating, black: copying), as calculated by the model

TABLE 2.9: A comparison of the three models' output on some MR of the E2E+ test set. The first reference utterance is reported for convenience

MR	`name[New Viet Huong], eatType[pub], customer rating[1 out of 5], near[Ecco]`
1st reference	The New Viet Huong is a pub near Ecco that has a customer rating of 1 out of 5.
EDA_CS	New Viet Huong is a pub near Ecco with a customer rating of 1 out of 5.
TGen	New Viet Huong is a pub near Ecco with a customer rating of 1 out of 5.
EDA	Near the riverside near the ERNick Restaurant is a pub near the ERNicker's.
MR	`name[La Mirabelle], eatType[restaurant], food[Iraqi], priceRange[high], area[riverside], familyFriendly[yes], near[Mi Cocina]`
1st reference	La Mirabelle is a children friendly restaurant located in the Riverside area near to the Mi Cocina. It serves Iraqi food and is in the high price range.
EDA_CS	La Mirabelle is a high priced Iraqi restaurant located in the riverside area near Mi Cocina. It is children friendly.
TGen	La Mirabelle is a high priced Iraqi restaurant in the riverside area near Mi Cocina. It is child friendly.
EDA	La Memaini is a high priced restaurant that serves Iranian food in the high price range. It is located in the riverside area near Manganaro's Restaurant.

We can therefore follow the white spots, corresponding to higher values of attention, to understand the flow of the model's attention during the generation of the output utterance.

Moreover, p_{gen} values, which lie in the numeric interval $[0, 1]$, help us in the interpretation of the Attention: they are represented as a grayscale vector from zero (black) to one (white) under the matrices. Values close to 0 mean copying and those near 1 mean generating.

We can note that our model's behavior varies significantly depending on the dataset it has been trained on. Figure 2.6a shows the attention probability distribution matrix of EDA_CS (together with p_{gen} vector) trained on the E2E dataset: as observed before, attribute values in this dataset have a very low variability (and are already present in the training set), so they can be individually represented and easily generated by the decoder. In this case, a typical pattern is the copy of only the first discriminating character, clearly noticeable in the graphical representation of the p_{gen} vector, and the subsequent generation of the rest of the word. Notice that the attention tends to remain improperly focused on the same character for more than one output time step, as in the first letter of "high".

On the other hand, the copy mechanism shows its full potential when the system must learn to copy attribute values, as in the E2E+ dataset. In Figure 2.6b the diagonal attention pattern is pervasive: (i) it occurs when the model actually copies, as in "Harley Davidson" and "Coco Pazzo", and (ii) as a *soft track* for the generation, as in "customer rating", where the copy-first-generate-rest behavior emerges again.

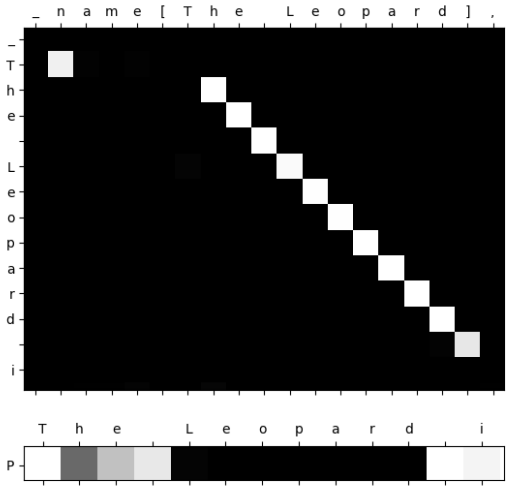


FIGURE 2.7: Copying common words leads the model to "uncertain" values of p_{gen}

A surprising effect is shown in Figure 2.7, when the model is expected to copy words that, instead, are usually generated: an initial difficulty in copying the word "The", usually a substring of a slot value, is overcome as follows. The first character is purely generated, as shown by the white color in the underlying vector, and the sequence of the following characters, "he_", is half-generated and half-copied. Then, the value of p_{gen} gets suddenly but correctly close to 0 (black) until the closing square bracket is met. The network's output is not affected negatively by this confusion and the attention matrix remains quite well-formed.

As a final remark on the used metrics: while being useful, well-known and broadly accepted, they do not perfectly reflect the ability to directly copy input facts to produce outputs, so settling the rare word problem. New metrics that give greater importance to rare words might be needed in the future, with the purpose of better assess performances of able-to-copy NLG models on datasets such as the E2E+ one.

Surely, the architectural updates and the peculiar training pipeline of EDA_CS make our model training time per epoch slower than EDA and EDA_C. One training epoch on EDA lasts for $\sim$ 30 minutes, $\sim$ 37 for EDA_C and $\sim$ 84 for EDA_CS. Even though this may sound like an issue, in practice our model needs far less epochs to train to similar loss values (and consequently output quality) than the other two architectures. In Fig. 2.8 we can see the training loss curves of the 3 before-mentioned models on the E2E+ dataset; the red dashed line indicates a level of equal loss at 3.5, which we found to be a reasonable training loss after which the models tend to overfit. The plot shows how our model reaches good loss values (the red dashed line) already at the end of the second epoch, while EDA_C and EDA takes almost 4 and 14 times longer respectively. This is another evidence of the fact that our model is actually very efficient in squeezing information from the data.

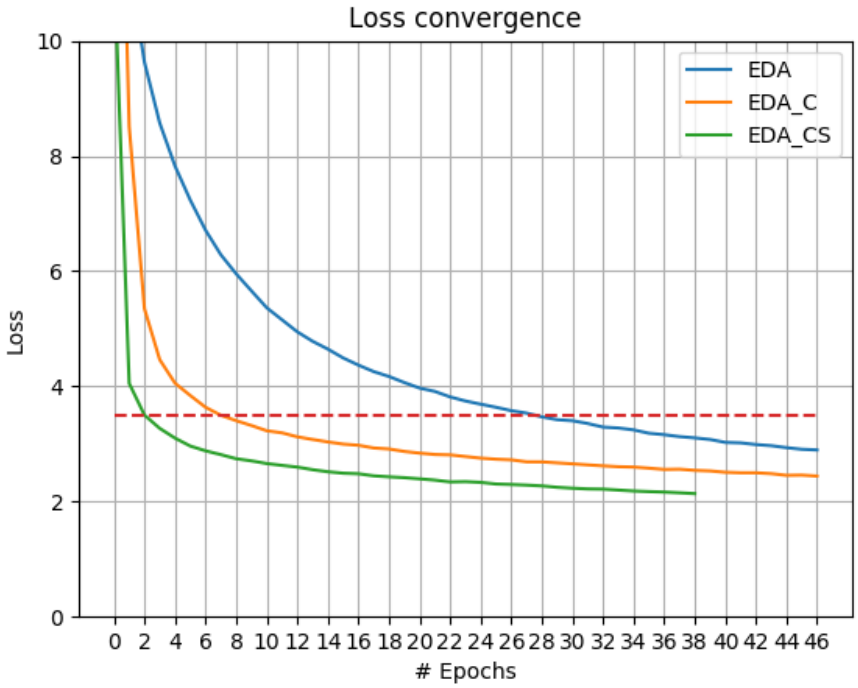


FIGURE 2.8: Comparison of training losses of EDA, EDA_C and EDA_CS on the E2E+ dataset.

Chapter 3

A Retrieval-Augmented Approach for more Informative and Accurate Dialog Generation

Automatic dialog generation has rapidly grown in the last few years (S. Zhang et al., 2018), becaming a fundamental component for many real-world, challenging applications, such as virtual assistants, conversational agents (Tudor Car et al., 2020), advanced question answering applications (D. Guo et al., 2018), or completion tools (Kannan et al., 2016), and is also a matter of great concern for companies and organizations relying on artificial intelligence solutions to enhance millions of daily interactions through their services.

Simple single-turn Seq2Seq architectures, initially proposed for this task, often fail to capture long-term temporal dependencies across dialog turns (J. Li et al., 2016; Sutskever et al., 2014; Vinyals & Le, 2015).

Multi-turn Seq2Seq models, such as the hierarchical recurrent encoder decoder (HRED) (I. Serban et al., 2016; I. V. Serban et al., 2017; Xing et al., 2018) have tried to alleviate these problems, yielding responses more coherent with the dialog contexts. Nonetheless, the generated texts tend to be either generic or too short, and not comparable with the human ones.

Recently, pretrained transformer-based models such as BERT (Devlin et al., 2018), Transformer-XL (Dai et al., 2019), XLNet (Yang et al., 2019) and ERNIE (Z. Zhang et al., 2019) led to state-of-the-art performance on many Natural Language Processing/Understanding tasks, including Question Answering, Sentence Classification, Sentence Similarity Inference, Named Entity Recognition, etc. The architectures of two prominent examples of such models are described in sec. 3.2.2 and in sec. 3.2.3 for the Transformer and the Transformer-XL respectively.

An interesting idea for further enhancing a generative model performance is to condition the generation on samples retrieved from a task-related datastore.

The model we present in this chapter (Bonetta et al., 2021a) exploits a similar framework; our first original contribution concerns how to augment a Language Model (LM) with a k-Nearest Neighbors (kNN) and a generative module in order to use it for actual text generation. Furthermore, in the dialog generation framework, the typical dialog structure is used to enhance and speed up the retrieval mechanism, improving the generation results.

In Section 3.1 a brief overview of some related works is presented. Section 3.2 outlines the self-attention mechanism and the main transformer-based architectures. In Section 3.3 we describe our model, formally defining our approach, and also going into detail of the retrieval mechanism. The remaining sections are devoted to the dataset descriptions and results discussion.

3.1 Related Works

The approaches we deal with here focus on incorporating external knowledge in the generation process, both explicitly or implicitly.

3.1.1 Retrieval Augmented Models

Retrieval augmented models typically rely on a datastore to retrieve sensible explicit information, like piece of texts, to further condition the generative process. (Guu et al., 2020; K. Lee et al., 2019) augment a generative model with a neural retriever trained to pick informative text paragraphs. Another approach has been used in (Guu et al., 2017) where they show how to retrieve prototype sentences from the training corpus and then refine them to get better generative outputs. (Khandelwal et al., 2020) propose to interpolate a LM with a KNN system. At every time step the LM outputs a distribution of probabilities over a vocabulary and the kNN-system outputs distances over k-nearest-neighbors tokens from a datastore. Such distances can be normalized to have a probability distribution over the KNNs, and finally the two distribution can be interpolated. Anther approach (Karpukhin et al., 2020) consists in training models to minimize the distances between context and continuation embeddings in a hidden space. LM or masked LM are used to generate such embeddings.

3.1.2 Memory Augmented Models

There have been some other works about learning mechanisms which read from memories, typically through attention heads, (Sukhbaatar et al., 2015; Weston et al., 2014) and incorporate these techniques to enhance text generation models (Z. Lin et al., 2019; Madotto et al., 2018). Although these approaches successfully integrated systems which can retrieve and even update information from memory stores, it is non trivial to scale these ideas to millions or more memories. It is also difficult to directly interpret retrieved memories, which are not grounded to some specific resources, like text spans.

3.2 Self-Attention Models

As we have described in chapter 2 RNN models, either implemented via LSTMs or GRUs have been widely used in the DTT field and more generally in NLP related tasks (J. Li et al., 2022; M. Luong et al., 2016; T. Luong et al., 2015; Ranzato et al., 2016) with great success. This has been made possible by the advent of the Encoder-Decoder architectures paired with the attention mechanisms which, although being effective, show some drawbacks mainly due to their sequential nature. These models usually unroll computation factorizing the output probabilities along the input or output sequences, and use attention

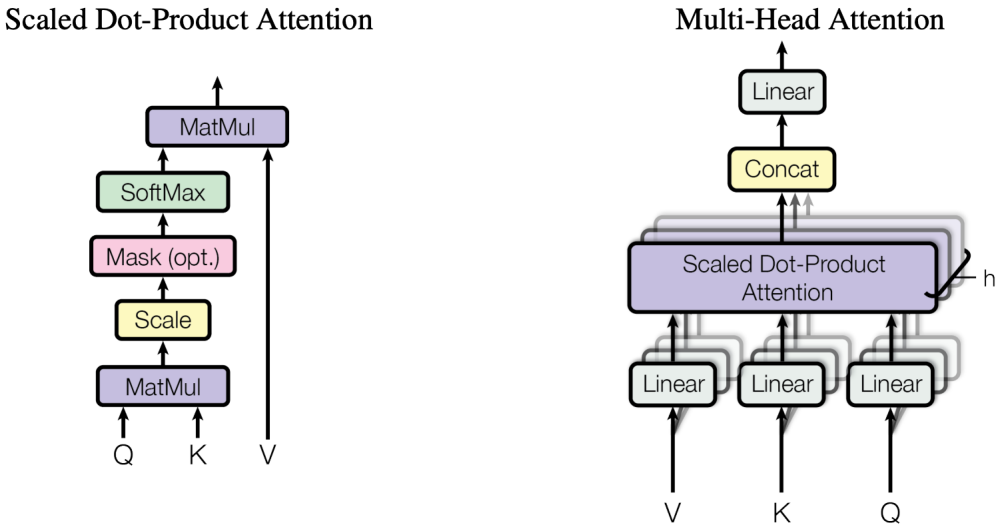


FIGURE 3.1: graph representing the scaled dot-product self-attention on the left, and its multi-head version on the right[1].

.

to align symbols with time-steps. This force the network to define a temporal order in the data since every piece of computation done at a fixed time-step is conditioned on the previous one.

As a consequence, it is difficult (or even impossible) to parallelize computations within training samples, specially when using a deep model, since every RNN layer needs to wait for results from previous time-steps and from previous layers. Moreover, these architectures creates computational graphs where the number of operations required to relate different arbitrary input or output positions grows by the distance between them. The resulting issue is that the model tends to learn patterns which relate inputs close to themselves (Pascanu et al., 2013). If we look for example at how word embeddings are created by the encoder stack in a Encoder-Decoder architecture we see that the aforementioned issue leads to sub-optimal word embeddings since there is no simple way to create such embeddings injecting information from very distant tokens.

The Transformer model (Vaswani et al., 2017) has been created with the intention of solving these problems and create a more efficient, prallelizable and explainable Encoder-Decoder architecture. Its basic core-idea is to avoid any recurrence in the model and to use an attention mechanism, called self-attention, as principal computational block.

3.2.1 The Self-Attention

The self-attention mechanism described in the Transformer (see fig. 3.2 for a view of the model architecture) takes inspiration from the original attention mechanism for Encoder-Decoder architectures (Bahdanau et al., 2015) and from other subsequent works like (Cheng et al., 2016; A. Parikh et al., 2016) where the so called intra-attention mechanism is proposed. These techniques have the specific benefit to uncover patterns and relations among tokens in the same sequence.

[1]Figure from (Vaswani et al., 2017)

TABLE 3.1: Different approaches for computing the attention alignment. q and k_i are the query and key respectively. *sim* is a similarity functions such as cosine. W_{imp}, W_x and b are trainable parameters.

Name	Reference	Formula
Additive	Bahdanau et al. (2015)	$b^T \cdot \tanh(W \cdot [q;h])$
General	T. Luong et al. (2015)	$q^T W k_i$
Dot-product	T. Luong et al. (2015)	$q \cdot h$
Scaled dot-product	Vaswani et al. (2017)	$\frac{q \cdot h}{\sqrt{emb}}$
Similarity	Graves et al. (2014)	$sim(k_i, q)$
biased general	Sordoni et al. (2016)	$k_i(Wq + b)$
activated general	Ma et al. (2017)	$act(q^T W k_i + b)$
generalized kernel	Choromanski et al. (2021)	$\phi(q)^T \phi(k_i)$
concat	T. Luong et al. (2015)	$W_{imp}^T act(W[q;k_i] + b)$
additive	Bahdanau et al. (2015)	$W_{imp}^T act(W_1 q + W_2 k_i + b)$
feature-based	Y. Li et al. (2019)	$W_{imp}^T act(W_1 \phi_1(K) + W_2 \phi_2(K) + b)$

The self-attention (see left side of fig.3.1) is a function that maps a query Q and a set of key-value pairs (K, V) to an output using the following formula:

$$Attention(Q, K, V) = Softmax(a(Q, K))V \tag{3.1}$$

$$a(Q, K) = \frac{QK^T}{\sqrt{d_k}} \tag{3.2}$$

where $a(Q, K)$ is called the alignment function since it relates queries and keys. $Q \in \mathbb{R}^{T_x \times d_k}$, $K \in \mathbb{R}^{T_x \times d_k}$ and $V \in \mathbb{R}^{T_x \times d_k}$ are linear transformations of the self-attention layer input, T_x is the length of such input and d_k is the key vector dimension. The specific attention alignment in Eq. 3.2 is called *scaled dot-product attention* since it computes a dot-product which is further scaled by d_k (left in figure 3.1).

Nowadays many variants of the alignment function are used; we show some of the most relevant implementations in table 3.1, but Eq. 3.2 remains one of the most widely used since it is very efficient to compute in modern hardware like GPUs and does not add extra weights to the network.

3.2.2 The Multi-Head Self-Attention and The Transformer

The multi-head self-attention (see Fig. 3.1 on the right) is a linear combination of H independently computed scaled dot-product attentions (see Table 3.1), whose queries, keys and values are linear projections of q, k, and v:

$$\text{multiHead}(q, k, v) = [head_1; \ldots; head_H] \cdot W_o \tag{3.3}$$

$$head_i = \text{attention}(q \cdot W_i^q, k \cdot W_i^k, v \cdot W_i^v), \quad i = 1, \ldots, H \tag{3.4}$$

Named *emb* the embedding dimension, $W_o \in \mathbb{R}^{emb \times emb}$ and, for each i, $W_i^q \in \mathbb{R}^{emb \times \frac{emb}{H}}$, $W_i^k \in \mathbb{R}^{emb \times \frac{emb}{H}}$, and $W_i^v \in \mathbb{R}^{emb \times \frac{emb}{H}}$. The number of heads H is chosen so that $emb \mod H = 0$.

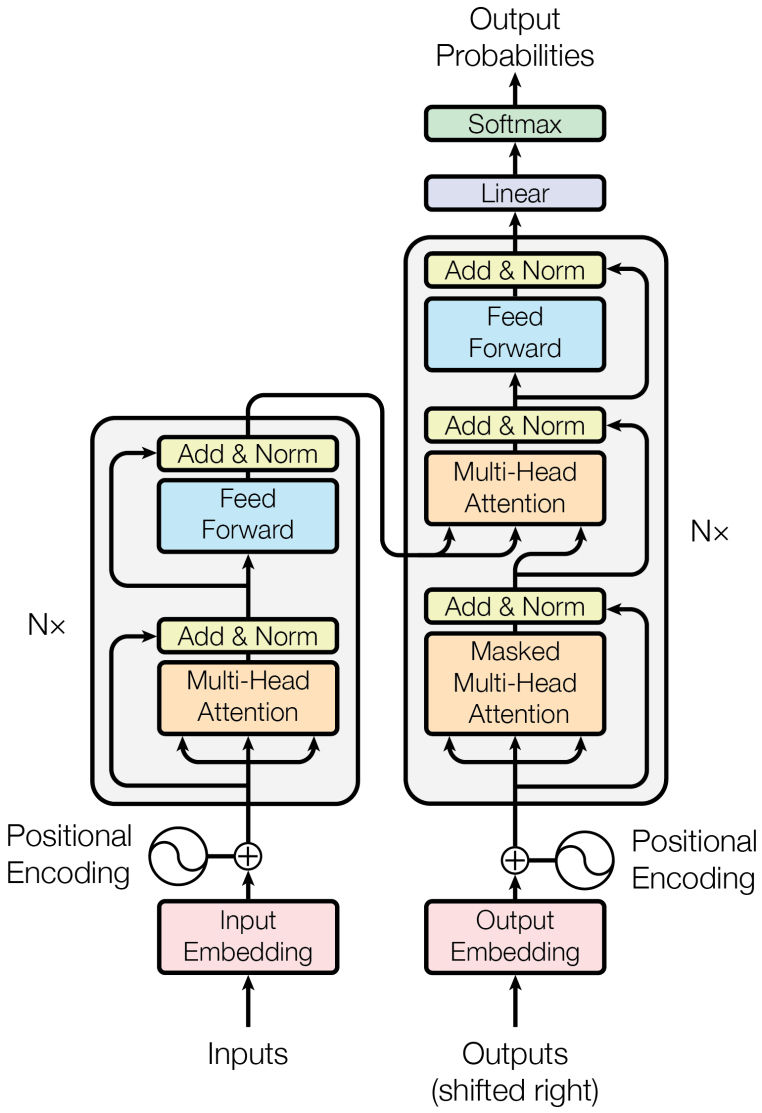


FIGURE 3.2: The Transformer architecture (figure from Vaswani et al. (2017)).

The feed-forward layer on top of each multi-head attention block in figure 3.2 is a single hidden layer perceptron which uses the Rectified Linear Unit (ReLU) activation function (Glorot et al., 2011; Jarrett et al., 2009; Nair & Hinton, 2010) and has f hidden neurons:

$$\text{FeedForward}(x) = \text{ReLU}(x \cdot W_1 + b_1) \cdot W_2 + b_2, \tag{3.5}$$

where $W_1 \in \mathbb{R}^{emb \times f}$, $b_1 \in \mathbb{R}^f$, $W_2 \in \mathbb{R}^{f \times emb}$, $b_2 \in \mathbb{R}^{emb}$ and $x \in \mathbb{R}^{T_x \times emb}$ is a generic input. Multi-head attention and feed-forward layers are followed by a residual connection (K. He et al., 2016) and layer normalization (Ba et al., 2016).

The Transformer is composed of two submodules, as can be seen in Figure 3.2:

- the *encoder* is made up of N identical stacked modules, each one consisting of a multi-head self-attention followed by a feed-forward layer (see Equation 3.5);
- the *decoder* is made up of N identical stacked modules, each one consisting of a multi-head self-attention and a multi-head input-output attention where the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. On top of each stacked module there is a feed-forward layer (see Equation 3.5).

Both input and output embeddings are enriched by a fixed, sinusoidal-based positional encoding (Gehring et al., 2017). Finally, a linear projection and a softmax activation determine the categorical distribution from which the t-th output token is sampled, as in Equation 2.3.

The transformer architecture has some important advantages over the RNNs:

- the amount of computation that can be parallelized, as measured by the minimum number of sequential operations required.
- the shorter path length among long-range dependencies in the network.

Learning long-range dependencies is a key challenge in many sequence transduction tasks. One key factor affecting the ability to learn such dependencies is the length of the paths that forward and backward signals have to traverse in the network. The shorter these paths between any combination of positions in the input and output sequences are, the easier the learning of long-range dependencies. Self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires $O(n)$ sequential operations. In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence length n is smaller than the representation dimensionality *emb*, which is most often the case with sentence representations used by state-of-the-art models in machine translations, such as word-piece (Wu et al., 2016) and byte-pair (Gage, 1994) representations.

3.2.3 The Transformer-XL

In this section we introduce the Transformer-XL (i.e. Transformer extra-long) (Dai et al., 2019) which, as the name suggests, takes inspiration form the Transformer and adapt it to deal with very long sequences. In the next Section 3.3

we will show how to updat it for sequence to sequence modeling and how to add a retrieval mechanism in order to improve generation even further.

The Transformer-XL is no more an Encoder-Decoder model, but relies on an upgraded version of a Transformer's decoder block and has been released for language modeling purposes, even though not for language generation per se. This means that it cannot be used off the shelf to learn sequence to sequence tasks but it shows some impressive results when used to compute the conditioned probability of a given piece of text or to create context representations. This model reads inputs left to right and computes the probability of a sequence $x_1, x_2, ..., x_n$ maximizing the objective:

$$P(x_1, x_2, ..., x_n) = \prod_{n=1}^{N} P(x_n | x_{<n})$$

In order to effectively encode an arbitrarily long context into a fixed size representation a simple solution would be to process the entire context sequence using an unconditional Transformer's decoder. This solution, even though simple in the idea, is not easily scalable to long inputs since the memory and computational requirements of the self-attention modules scales more than linearly in the input length. To address these limitations the Transformer-XL implements two ideas:

- Recurrence mechanism. It adds a recurrence mechanism to the Transformer architecture. During training, the hidden state sequence computed for a given chunk of text is fixed and cached (i.e. creating *memories*) to be reused as an extended context when the model processes the next chunk. Although the gradient still remains within a segment, this additional input allows the network to exploit information in the history, leading to an ability of modeling longer-term dependency and avoiding context fragmentation. This process is similar to truncated back propagation (Werbos, 1990).
- Relative Positional Encodings. Instead of incorporating positional bias statically into the initial embedding as happens in the Transformer, one can inject the same information into the attention score of each layer. More importantly, it is more intuitive and generalizable to define the positional bias in a relative manner. For instance, when a query vector q_i attends on the key vectors $k_{\leq i}$, it does not need to know the absolute position of each key vector to identify the order of the segment. Instead, for q_i it is sufficient to know the relative distance between each key vector $k_{\leq j}$ and itself, i.e. $i - j$. Practically, one can create a set of relative positional encodings $T \in R^{L_{max} \times d}$, where the i-th row indicates a relative distance of i between two positions and T is the maximum allowed distance. By injecting the relative distance dynamically into the attention score, the query vector can easily distinguish the representations, making the state reuse mechanism feasible.

TABLE 3.2: Dataset specifications.

	Taskmaster-1	**Prop. dataset**
# dialogs	7,708	1,328,301
# turns	169,467	21,953,321
# unique tokens	29,626	1,601,647
avg. turn per chat	21.99	16.53
avg. tokens per turn	7.83	18.00

3.3 Retrieval Augmentation for Dialog Generation

In this section our method is outlined (Bonetta et al., 2021a), which improves dialog generation by exploiting memorized information from the training data without further model training. At inference, turn generation is enhanced by interpolating the next word distribution based on the trained LM with the one based on a kNN search system. A single LM forward pass over the training data is preliminary conducted to compute context-target pairs and store them in a key-value pair *datastore*, which will be queried to perform the kNN search.

The rest of the section is organised as follows: in the next Section 3.3.1 the datasets used for experimentation are presented. In Section 3.3.2 the process for building a datastore is described, and Section 3.3.3 shows how a kNN distribution is computed and used to augment the LM.

3.3.1 Datasets Description

Two different datasets are used as benchmarks for our method: a public dataset, the Taskmaster-1 and a real, company collected, call center customer service dataset. Both datasets contains dialogs between users and agents where the agent role is to help users with their needs, answering specific questions, providing services or instructing users about different product possibilities. Since these chats are driven by some user's need, these datasets are know as *task-oriented* or *goal-oriented* datasets.

Taskmaster-1 dataset. Taskmaster-1 (Byrne et al., 2019) is a crowsurced dataset, released by Google in 2019, where Amazon turkers were asked to write dyadic dialogs following some given set of instructions describing six tasks: ordering pizza, creating auto repair appointments, setting up rides for hire, ordering movie tickets, ordering coffee drinks and making restaurant reservations. Workers were asked to play the role of both assistant and user. Specifically, they were told to write a scenario in which they are speaking to their assistant on the phone while the assistant accesses the services for one of the given tasks.

The dataset is composed of 13,215 task-based dialogs (12,355 for the trainingset and 770 for the test set), including 5,507 spoken and 7,708 written dialogs. The input is also called *context* and contains agent-user utterances up to a user one; the following agent utterance is the target. Note how some special tokens (i.e. <user>, <enduser>, <agent>, <endagent>) are used to bound each utterance; these delimiters are useful for the network to understand who is talking. A Taskmaster-1 sample is shown in table 3.3; more info about the dataset are in table 3.2.

TABLE 3.3: A Taskmaster-1 sample.

Input
<user>Hi there, could you please help me with an order of Pizza?<enduser> <agent>Sure, where would you like to order you pizza from?<endagent> <user>I would like to order a pizza from Domino's.<enduser> <agent>What kind of pizza do you want to order? <endagent> <user>What are the specials they have right now? <enduser> <agent>There are family and party combos currently on offer <endagent > <user>No, I just want a large pizza <enduser> <agent>They have any large specialty pizza for 10.99 <endagent> <user>What are their specialty pizzas? <enduser>
Target
<agent>Well, there is the Extravagazza, Meatzza, Philly Cheesesteak, Hawaiian, Buffalo Chicken Ranch, and more. Would you like to hear more? <endagent>

Proprietary (Prop.) dataset.[2] This dataset contains dyadic agent-user chats collected from a financial service call center over a one year time period, giving us the opportunity to test our approach in a real company scenario. It contains 172 times the dialogs number of the Taskmaster-1, as shown in table 3.2, and comes with two meta-information for every utterance, the *turn numbers* and the *agent-ids*. The turn number is just the position of the specific turn within the chat. The agent-id is a unique identifier for the agent speaking and usually it is just its email address. These meta-information help the network to discern different moments in the chats (like chat beginning/middle/end) and ideally to model every agent's style and nuances in how different agents handle conversations. Meta-information are concatenated to the chat's text as prefixes to every turn, following the approach used in (Wolf et al., 2019). An example is given in table 3.4.

TABLE 3.4: A proprietary dataset sample.

Input
<agent> *1 Rob@company* | Hi! What questions can I help answer about our products and services? <endagent> <user> *1* | I would want to use the account by the end of the week . So should I go into a branch office to sign the signature card? <enduser> <agent> *2 Rob@company* | Before we get started may I please have your name and the name of you business? <endagent> <user> *2* Tony Stark; Stark Industries <enduser>.
Target
<agent> *3 Rob@company* | Hello and welcome! I'd be happy to assist you. <endagent>

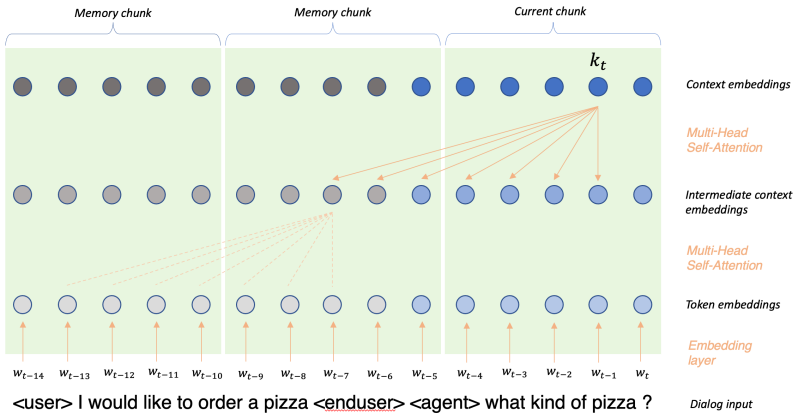


FIGURE 3.3: Transformer-XL workflow. Given a dialog the model processes it chunk by chunk from left to right creating the dialog context embeddings.

3.3.2 Datastore Creation

The *datastore* is the source of information that allows the network to be more precise during inference, lowering the model uncertainty about the next-token distribution. A datastore contains context embeddings and the first token which follows each context. Usually they are gathered from a training subset, even though also other possibilities can be considered, like using another dataset with a similar distribution to the training one. How to use this source of information is better described in the next Section 3.3.3; here instead follows a more precise description of what a datastore is made of and how to build it.

Let $(c_t^i, w_t^i) \in D$ be the i^{th} example in training data D. The context c_t^i is a sequence of dialog turns of a dyadic chat occurring between an agent and a user; c_t^i is represented as a sequence of tokens, i.e. $c_t^i = (w_1^i, w_2^i \dots w_{t-1}^i)$, and w_t^i is the target word.

Let $f(c_t^i)$ denote the context-encoder function, that maps the context c_t^i to its fixed-length vector embedding. More specifically, $f(c_t^i)$ represents the embedding of token w_{t-1}^i after attending to all the previous tokens in the example. We define $f(\cdot)$ as the input to the last feedforward layer in the final attention block of a Transformer-XL (see 3.2.3), as in (Khandelwal et al., 2020). This achieves better performance than other options (e.g, the output of the last transformer layer). In order to have a sensible embedding function $f(\cdot)$ we need to train the Transformer-XL on the training data. This is achieved by cross entropy minimization and controlling overfitting through well known techniques like early stopping on validation data performance. Differently from (Dai et al., 2019) and (Khandelwal et al., 2020), which train a LM by concatenating all the examples, we train the model by resetting the Transformer-XL states at the beginning of each chat: this effectively prevents the model from conditioning on previous unrelated contexts. Moreover, memories are used to propagate information inside very long chats.

Finally, the trained LM is used to build the datastore (K, W) through one forward pass on the training set, computing the embeddings of all the contexts

[2]The dataset can not be made public due to privacy constraints

within it:

$$(K, W) := (k_t^i, w_t^i) = (f(c_t^i), w_t^i), \quad \forall (c_t^i, w_t^i) \in D$$

where $k_t^i = f(c_t^i)$ is the vector representation of the context, and w_t^i is the target word id (i.e. integer number). Note that every context embedding is just the embedding of its last token, and since we consider every possible context in the training set, the datastore contains one embedding for each token within it. In Fig. 3.3 it is shown an intuitive representation of the model workflow. Given a dialog, it processes it chunk by chunk using the previous ones as memories. The multi-head self attention mechanism is applied (2 times in the figure) to generate the context embedding k_t.

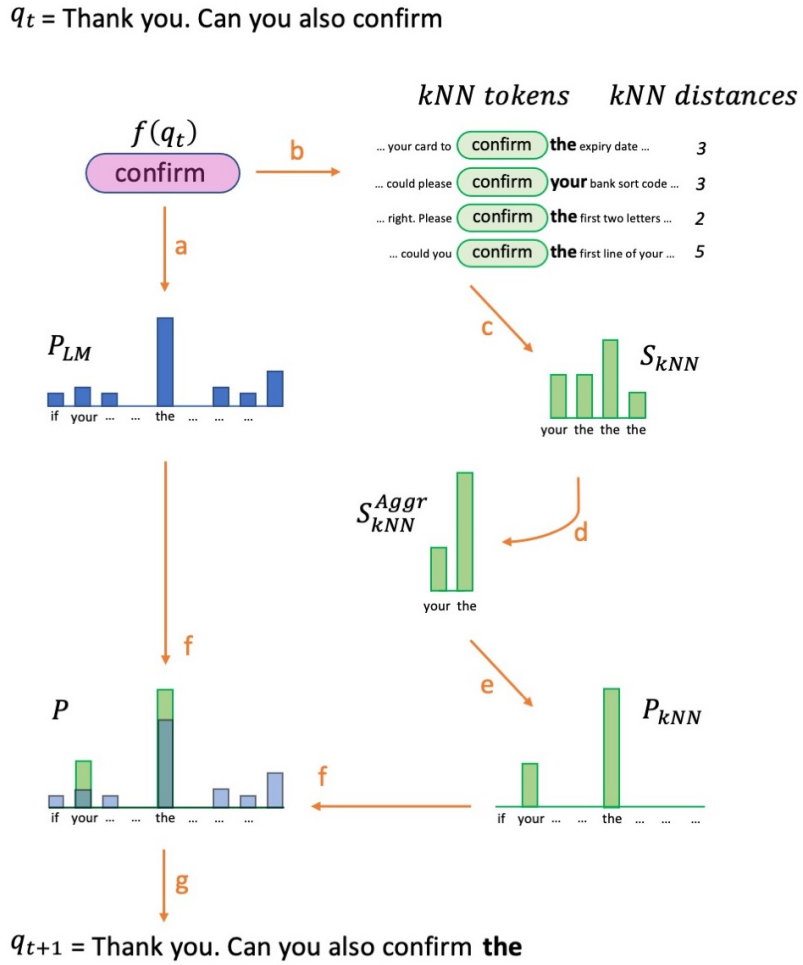


FIGURE 3.4: Illustration of the generation process. Given a context q_t, its embedding is computed with the pretrained Transformer-XL and the same model generates a probability distribution on the next word P_{LM}. The embedding is used to query the index (i.e. the datastore) for similar contexts. A distribution P_{kNN} is built using the words that come next to the retrieved contexts; the closer the retrieved context to q_t the higher the word probability. Finally P_{kNN} and P_{LM} are interpolated and the next token is sampled from P.

3.3.3 Hybrid Probability Distribution

Here we see how the actual information from the datastore can be used at inference time to improve the generative capabilities of a typical LM. At every time step t, the trained LM receives a *query* (q_t), i.e. a chat truncated at the end of a user turn, and generates the next assistant turn token-by-token, according to the following steps, also illustrated in Fig. 3.4:

a) Generate the context embedding $f(q_t)$ and the probability distribution $P_{LM}(v_t|q_t)$ over next words in the vocabulary.

b) Issue a kNN search with $f(q_t)$ as query, to get from the datastore its nearest neighbors N_t:

$$N_t = \{(k_1, w_1), (k_2, w_2) \dots (k_n, w_n) \dots\}$$

c) Compute the score $S_{kNN}(w_n|q_t)$ of the token w_n over N_t, based on L^2 distance between k_n and $f(q_t)$:

$$S_{kNN}(w_n|q_t) = \frac{e^{-d(k_n, f(q_t))}}{\sum_{k_j \in N_t} e^{-d(k_j, f(q_t))}}$$

d) Aggregate the scores of each vocabulary token w_n as the sum of all its occurrences within the retrieved neighbors:

$$S^{Aggr}_{kNN}(w_n|q_t) = \sum_{w_{n'} \in N_t w_{n'} = w_n} S_{kNN}(w_{n'}|q_t)$$

e) Get the probability distribution P_{kNN} over next words in the vocabulary:

$$P_{kNN}(v_t|q_t) = \sum_{(k_n, w_n) \in N_t} \mathbf{1}_{v_t = w_n} (S^{Aggr}_{kNN}(w_n|q_t))$$

where $\mathbf{1}_{v_t = w_n}$ is a vector whose dimension is equal to the vocabulary size and whose elements are all zero except for the t-th one, equal to 1.

f) Interpolate P_{kNN} with P_{LM}, using a weighted sum based on the interpolation hyperparameter λ, to get the final probability distribution P for next word v_t :

$$P(v_t|q_t) = \lambda P_{kNN}(v_t|q_t) + (1 - \lambda) P_{LM}(v_t|q_t)$$

g) output the next word $\hat{v}_t$ by sampling from $P(v_t|q_t)$ and concatenate $\hat{v}_t$ to q_t to update the context: $q_{t+1} = q_t + \hat{v}_t$. If $\hat{v}_t$ is a terminal token the generation process stops; otherwise the entire procedure is repeated.

Since the Transformer-XL was built for Language Modeling purposes it lacks a generation module that actually samples from $P(v_t|q_t)$ and feed back the token to the model in an autoregressive way, so we implemented this module for greedy and beam search decoding.

3.3.4 Retrieval Mechanism

To search the datastore, we use FAISS (Johnson et al., 2017), an optimized open source library for fast nearest neighbor retrieval in high dimensional space. FAISS's central building block is the *index*, a structure which stores millions of key-value pairs for efficient search. An issue with the index is that the number of elements could easily grow to hundreds of millions, leading to memory issues and hindering the search performance. However in practice, we only need to store token embeddings for assistant turns, since we are only interested in generating assistant responses. So we propose the simple but effective idea of filtering out from the datastore every token coming from a user turn, so almost halving its size, and allowing the generation of consistent utterances, resembling assitant specific style. The indexes are all implemented as OPQ32,IVF8192_HWSW32,PQ32 as per FAISS index factory string [3]. This quite complex implementation provide a good tradeoff between search accuracy and speed by applying Optimized Product Quantization (Ge et al., 2013) as a pre-processing step and Product Quantization (Gray & Neuhoff, 1998) as post-processing, while using inverted lists to define cluster of vectors and applying Hierarchical Navigable Small World indexing (Malkov & Yashunin, 2020).

3.4 Experiments

In this section the implementation details of our model *Transformer-XL + kNN* are presented along with the results obtained for both datasets. We coded the system using the PyTorch framework release 1.2.0, and FAISS release 1.6.1. Training and inference were done using a single Nvidia Tesla V100 32GB GPU.

3.4.1 Transformer-XL + kNN on Taskmaster-1

For the Taskmaster-1 we used a Transformer-XL model with 12 layers, 8 heads, 512-dimensional hidden states and 2048 as inner attention dimension, resulting in 49M weights and trained for a maximum of 10k steps via Adam optimizer. The training stopping criterion is based on perplexity on the development set. Hyperparameter tuning, including optimal interpolation parameter λ determination, is done through performance evaluation over the development set. We adopted a BPE vocabulary consisting of 16K tokens generated using the Sentencepiece library (Kudo & Richardson, 2018). All the training set is used to build the datastore.

Our model is compared with two baselines: the Transformer, which is the best performing model proposed for this dataset by (Byrne et al., 2019); and the Transformer-XL, i.e. the LM used without the retrieval mechanism.

The first column of table 3.5 shows the corresponding BLEU scores[4], obtained as mean values of 10 different runs, and standard deviations. We can see that our method gets more than two BLEU points over the Transformer baseline, and more than one point over the Transformer-XL baseline. This result show

[3] more information on different FAISS indexes at https://github.com/facebookresearch/faiss/wiki/The-index-factory.

[4] BLEU script at: https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/bin/t2t-bleu

TABLE 3.5: Average BLEU and standard deviations on test set. The statistical significance is validated via Student's *t*-test with significance level of 99.8%.

	Taskmaster-1		**Prop. dataset**	
Models:	Avg	Std	Avg	Std
Transformer (Byrne et al., 2019)	6.11[5]	-	-	-
Transformer-XL	7.09	0.14	39.96	0.36
Transformer-XL + kNN	**8.30**	0.05	**41.72**	0.20

TABLE 3.6: Example of inference query, along with results from baseline and our best model. *agent@company.com* is the agent-id, which is preceded by the turn number. Tokens between angular parenthesis indicate the beginning and end of turns.

Query	
...	
<agent> *3 Luke@company* | I am thrilled to hear that you would like to apply for a new card today. Have you decided which card you would like to apply for?<agent>	
<user> *3* | yes <enduser>	
<agent> *4 Luke@company* | Excellent! <endagent>	
<user> *4* | basic credit card.<enduser>	
Target	
<agent> *5 Luke@company* | Our credit card is a great card for e-shopping and balance transfer. You can use the link below to begin the application today.<endagent>	
Transformer-XL	<agent> *5 Luke@company* | That is very good to know! <endagent>
Transformer-XL + kNN	<agent> *5 Luke@company* | That is a great card. I would be happy to stand by while you apply should you have additional questions. <endagent>

that the interpolation is effective in improving generative performance even if the datastore itself contains two order of magnitude less turns with respect to the proprietary dataset.

Figure 3.5 depicts the BLEU trend curve when the interpolation parameter λ varies through the selected range. We can see that kNN interpolation improves the BLEU scores over the Transformer-XL baseline for every value of λ in the selected range. The best result is with $\lambda = 0.4$, indicating LM and context retrieval are almost equally contributing.

3.4.2 Transformer-XL + kNN on a Real World Scenario

For the proprietary dataset we used the same model hyperparameters as for the Taskmaster-1 but augmented the hidden states dimension to 768 and the inner attention dimension to 3072, resulting in ~116M weights. We trained for a maximum of 400k steps.

Since using all the training set for the datastore would result in a prohibitively large disk space usage we decided to build it using just the last 3 months of the training set (1/4 of the entire data). This resulted in ~176M embeddings. Also in this case the Transformer-XL + kNN improves over the LM model for about 1.8 BLEU points, even with a datastore smaller then the entire training set. These results are obtained interpolating with $\lambda = 0.5$ (best on dev. set).

[5] result from referenced paper.

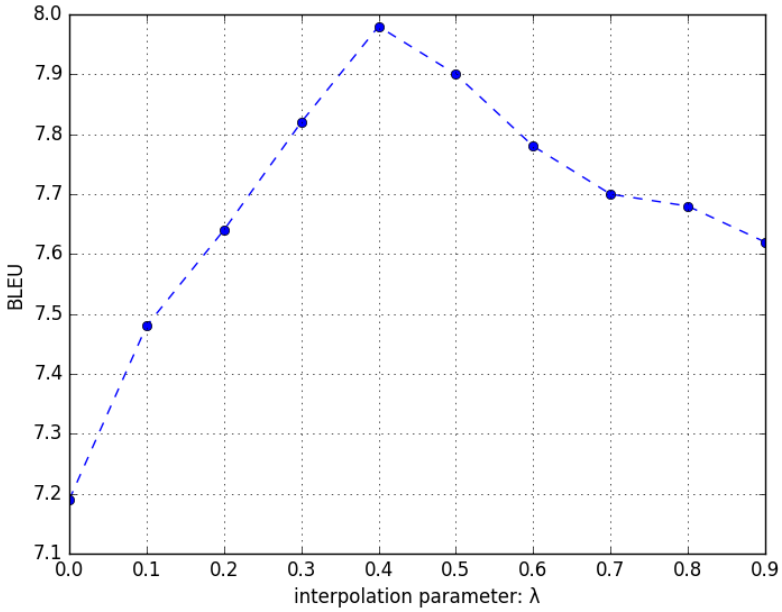


FIGURE 3.5: Taskmaster-1 BLEU trend at different λ interpolation values (development set).

Table 3.6 shows a sample from the test data along with the expected target, the turn generated by the Transformer-XL, and the turn generated by our Transformer + kNN. In this dialog a user wants some help for a credit card application. Our proposed model generates a sensible and relevant continuation: the agent conveys the intent to help the user apply for the credit card, as in the target. On the other hand the baseline Transformer-XL model generates a generic response which is not useful in advancing the dialog. The quality of our model has been assessed more thoroughly by human volunteers who evaluated the Transformer-XL + kNN outputs vs. a ranker model described in our patent (D. Liu et al., 2021). A description of the human evaluation and its findings is in Appendix A.2.

More experiments using different proprietary datasets and indexes are presented in Appendix A.1, where we assess the advantages of finetuning the model on subsets of the training set. Finally in Appendix A.3 is discussed a simple idea to constraint the system to generate train n-grams with the aim to have a more controllable and reliable generation process.

Chapter 4

Pruning Deep Language Generation Models

As outlined in the previous chapters, over the past few years deep learning models had been constantly establishing new state-of-the-art performance in a flood of different domains, including Language Generation (Dusek et al., 2020; Mei et al., 2016; Otter et al., 2018; Puduppully et al., 2019a), Machine Translation (Bahdanau et al., 2015; Vaswani et al., 2017), Image Processing (K. He et al., 2016; T. He et al., 2019; Simonyan & Zisserman, 2015; Szegedy et al., 2015; K. Zhang et al., 2020) and Image Captioning (Feng et al., 2019; L. Guo et al., 2020), just to name a few.

However, the resources required to properly train these systems can be prohibitive; in fact the number of weights of such neural networks can easily sum up to several millions. These growing performance costs have therefore induced scientists to look for techniques to limit the size of neural architecture parameters while keeping their powerful prediction properties.

An effective approach for reducing this complexity is via weights *sparsification*, which is the process that allows to achieve *sparsity* in the models' weights tensors, which in turn is the property that indicates that a subset of their values are set to zero. The major advantages of sparse models are:

- they have smaller computational requirements (specially on ad-hoc hardware) (Dietrich et al., 2021).
- they have smaller memory footprint compared to regular networks and can fit more easily in mobile devices.
- they are less prone to overfitting, potentially generalizing as well or even better than the original networks.

Numerous methods for inducing sparsity have been proposed over the past few years, and a non-exhaustive review of different approaches can be found in Section 4.1. Among them, those which are based on merely removing small norm weights, like via $L2$ regularization, are the simplest ones, and have demonstrated acceptable performances (Collins & Kohli, 2014; Han et al., 2015). $L2$ regularization based techniques, detailed in Section 4.1.3, add a penalty term to the cost functional in order to shrink parameters' values. All parameters dropping below a predefined threshold are then set to zero, thus obtaining sparse architectures.

A drawback of these methods is that neural weights' norms are all driven close to zero without taking account of weight relevance in the neural architecture, as discussed in detail in Tartaglione et al. (2018) for the Image Classification case.

The work we present in this chapter relies on this approach (Bonetta et al., 2022): we propose a new loss functional holding a suited regularization term, and demonstrate that application of stochastic gradient descent (SGD) algorithm allows to derive a new weights' update rule which selectively decreases the norm of non relevant weights, while performing a classic update for relevant ones. Weights' update therefore directly follows from loss optimization, not requiring the definition of ad hoc update rules, as frequently done in literature (Han et al., 2015; Tartaglione et al., 2018). Shrunk weights are then pruned in order to sparsify the neural architecture.

Our technique is general, as the proposed regularization term can be added to any loss functional regardless of its form, and constitutes a unified framework potentially exploitable for many different applications.

We verify the effectiveness of our method in the context of Natural Language Generation and Image Classification, respectively sparsifying self-attention based and convolutional neural architectures. While the latter task has been frequently addressed in literature, it is still rare to find sparsity techniques applied to language generation architectures. In particular we tested our method, inter alia, on the dialog generation task outlined in Chapter4.4.3.

Our approach establishes, at the best of our knowledge, new state-of-the-art performance for both the Language Generation tasks we dealt with, and gets comparable to state-of-the-art results in two out of four Image Classification tasks.

The Chapter is organised as follows: Sections 4.1 contains an overview of related works. Section 4.2 presents the theoretical foundations of our model and Section 4.3 the actual pruning algorithm. Section 4.4 describes datasets, implementation details and results obtained in the framework of Language Generation; implementation details and results concerning Image Classification are shown in appendix B.

4.1 Related Works

In literature there have been proposed many different techniques to prune neural networks. Here follows a discussion of some of the main ones, where algorithms which fall under similar theoretical frameworks are presented in the same subsections.

4.1.1 Dropout Methods

The idea behind dropout is to temporarily eliminate neuron/weight form the network with a preset probability. We distinguish two types of dropout techniques:

- Unit dropout randomly drops units (often referred to as neurons) at every training step to reduce dependence between units and prevent overfitting.
- Weight dropout randomly drops individual weights in the weight matrices at every training step. Intuitively, this connections dropping forces the network to adapt to a different connectivity at every training step.

Variational Sparse Dropout Variational Dropout (Blum et al., 2015) reformulates the learning problem with a Bayesian approach, where the dropout is modelled as noise applied to the inputs of all the neural layers, as follows:

$$\bar{g} = (\bar{x} \odot \Xi)\bar{w}, \tag{4.1}$$

where $\bar{x}$ is the input matrix, $\bar{w}$ is the weights matrix, Ξ is the noise matrix and $\bar{g}$ is the output matrix.

Given a drop rate α, the formulation in (Blum et al., 2015) applies on a weight $w_{i,j}$ a multiplicative Gaussian noise $\xi_{i,j} \sim N(1, p)$, where $p = \frac{\alpha}{1-\alpha}$.
This can be redrafted as:

$$\begin{gathered} w_{i,j} = \theta_{i,j}\left(1 + \sqrt{p} \cdot \epsilon_{i,j}\right) \sim \mathcal{N}\left(w_{i,j} \mid \theta_{i,j}, p\theta_{i,j}^2\right) \\ \epsilon_{i,j} \sim \mathcal{N}(0,1) \end{gathered} \tag{4.2}$$

where $\theta_{i,j}$ is the value of $w_{i,j}$ in the precedent iteration.

In the original paper (Blum et al., 2015) only $p \leq 1$ was taken into account, which implies a dropout rate $\alpha \leq 0.5$: unfortunately, this formulation generates a large variance of the gradient, caused by multiplicative noise, when $p = +\infty \implies \alpha = 1$,.
The reformulation given by the Variational Sparse Dropout (Molchanov et al., 2017) replaces the multiplicative factor with an exactly equivalent additive noise term:

$$\begin{gathered} w_{i,j} = \theta_{i,j}\left(1 + \sqrt{p} \cdot \epsilon_{i,j}\right) = \theta_{i,j} + \sigma_{i,j} \cdot \epsilon_{i,j} \\ \epsilon_{i,j} \sim \mathcal{N}(0,1) \end{gathered} \tag{4.3}$$

where $\sigma_{i,j}^2 = p \cdot \theta_{i,j}^2$ is treated as a new independent variable. This new reformulation solves the problem concerning the gradient, thus making it possible to use dropout values close to 1.

A regularization term in the loss, working in conjunction with the dropout, allows to reach very good results in terms of sparsification.

(Blum et al., 2015; Molchanov et al., 2017) are the first works in which dropout is used to sparsify networks, paving the way for new research in the field (Gomez et al., 2019; Salehinejad & Valaee, 2020)). Because of its easy implementation, this sparsification technique is used in very complex models, yielding optimal results (Gale et al., 2019) in many different tasks.

Targeted Dropout Targeted Dropout (Gomez et al., 2019) is another dropout-based approach, which focus on disentangling important subset of weights

from unimportant ones in order to get stable pruning. Before computing the gradients for each weight update, targeted dropout stochastically selects a set of units or weights to be dropped, using L1 and L2 norm as a proxy for weight importance, and then computes the gradients for the remaining weights. The authors use two importance criteria, one for unit pruning, the other for weight pruning and a targeting rate γ for select the bottom $\gamma|\theta|$ weights and neurons (i.e. the dropout candidates). Then the candidates are eliminated independently with a drop rate α. Targeted Dropout was tested on several known neural models, showing better performance than Variational Sparse Dropout.

4.1.2 Pruning using Weights' Weight

In this family we have pruning techniques that associate a value of relevance to the weights, based on their contribution in changing the output. The aim is to eliminate, in the pruning phase, all those weights with low relevance.

Single shot network pruning based on connection sensitivity (SNIP) The authors of SNIP (N. Lee et al., 2019) use a binary matrix $\bar{r}$ to represent the relevance of the weights. The relevance values of $\bar{r}$ are learned together with the value of the weights $\bar{w}$, in the backpropagation phase. The model will have twice as many parameters to be learned, but in a single step we can prune γ weights, where γ is a hyperparameter decided by the user. The SNIP algorithm proceeds as follows:

- Sample a mini-batch of training data.
- Calculate the weights relevance based on the magnitude of the derivatives.
- Order the weights by their relevance in descending order.
- Set $\bar{r}$ of the top-γ weights to 1, set $\bar{r}$ of the others weights at 0.
- Train the network with $\bar{r}$ applied.

The new optimization problem is defined as:

$$\begin{gathered}\min_{\bar{r},\bar{w}} L(\bar{r} \odot \bar{w} \mid \mathcal{D}) = \min_{\bar{r},\bar{w}} \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} L\left(\bar{r} \odot \bar{w} \mid (x_i, y_i)\right) \\ \text{s.t.} \quad \bar{r} \in \{0,1\}, \quad \|\bar{r}\|_0 \leq \gamma\end{gathered} \tag{4.4}$$

where $\mathcal{D}$ is the training dataset, x_i is the i-th input and y_i is the i-th expected output.

The strength and weakness of this algorithm is that a predetermined amount of weights are cut in a single step, resulting in models with limited sparsity or degraded performance. Despite this, SNIP has been the basis for new techniques (Y. Guo et al., 2016) which improved on this drawback.

Dynamic Network Surgery (DNS) According to the authors (Y. Guo et al., 2016), assigning a relevance to the weight is a very complicated task, because of the mutual influences and mutual activations among interconnected neurons.

In order to overcome this issue they propose a dynamic relevance calculation based on a binary relevance matrix $\bar{r}$ which, differently from (N. Lee et al., 2019), can change during training, as well as the weights matrix $\bar{w}$.
The DNS algorithm proceeds iteratively as follows:

- Perform the forward propagation and compute the loss with respect to $(\bar{r} \odot \bar{w})$, similarly to (4.4).
- Compute the backward pass from the model output.
- Update the relevancies $r^n_{i,j}(t+1)$, one by one, according to (4.5) where t is the time index and n the layer index. Once the new loss is calculated, all weights $\bar{w}$ are updated, even those with relevance equal to 0.

In the last step we see the dynamic property: after updating the weights, the mistakenly pruned connections are re-established if they appear to be important.
The matrix $\bar{r}$ is calculated from the weight matrix $\bar{w}$:

$$r^n_{i,j}(t+1) = \begin{cases} 0 & \text{if } \alpha > \left|w^n_{i,j}\right| \\ r^n_{i,j}(t) & \text{if } \alpha \leq \left|w^n_{i,j}\right| < \beta \\ 1 & \text{if } \beta \leq \left|w^n_{i,j}\right| \end{cases} \tag{4.5}$$

where α and β are two user-defined hyperparameters.

Because the results are very competitive in terms of pruning and performance, DNS is one of the most used weight relevance pruning techniques for comparison with new methods.
Nevertheless, when dealing with very large models, other approaches are preferred, since pruning techniques based on weight relevance cause the model doubles in size, because it is necessary to train also the relevance matrix $\bar{r}$.

4.1.3 Regularization Techniques

Regularization methods turn an original unstable, ill-posed problem into a well-posed one (Badeva & Morozov, 1991); they limit the capacity of models by adding a parameter norm penalty to the loss functional L, to control overfitting and decrease the test error.

Adding a regularization term to the loss allows to obtain interesting sparsification properties (Han et al., 2015). The new minimization problem becomes:

$$\hat{L}(\theta) = L(\theta) + \lambda F(\theta), \tag{4.6}$$

where L is any measure of Loss (like MSE, Cross Entropy) and λ is a hyperparameter that controls the contribution of the regularization term.

*L*0 **Regularization** *L*0 regularization when applied to neural weights $w^n_{i,j}$ is characterized by a factor $F(\theta)$ that penalizes all weights different from 0:

$$F(\theta) \equiv F(\bar{w}) = \sum_{n,i,j} \mathbb{I}\left[w^n_{i,j} \neq 0\right] \tag{4.7}$$

$w^n_{i,j}$ indicates the j-th connection of the i-th neuron in layer n and w indicates the weight matrix.
Unfortunately, the Loss is non-differentiable and intractable, due to combinatorial nature of the 2^θ possible states of the parameter θ.
In (Louizos et al., 2018) the problem is reformulated using a Bernullian distribution and a hard-sigmoid to make it tractable, achieving interesting results in terms of sparsity and performance.

Differently from *L*1 and *L*2 regularization, discussed in the following section, *L*0 is very expensive from a computational point of view: moreover, the implementation is relatively complicated and difficult to generalize. Nevertheless, *L*0 has been used in several models, even very complex ones, obtaining good results (Gale et al., 2019).

*L*1 **and** *L*2 **Regularization** *L*1 regularization exploits a regularization term of the form

$$F(\theta) = \sum_{n,i,j} \left|w^n_{i,j}\right|. \tag{4.8}$$

*L*2 regularization, one of the most common parameter norm penalty also known as Tikhonov regularization, ridge regression or weight decay (Tikhonov, 1963; Tikhonov & Arsenin, 1977), uses a quadratic regularization term:

$$F(\theta) = \sum_{n,i,j} \left|w^n_{i,j}\right|^2 \tag{4.9}$$

Substituting (4.9) in (4.6) the regularized loss L becomes:

$$\tilde{L}(\bar{w}) = L(\bar{w}) + \lambda\|\bar{w}\|^2 = L(\bar{w}) + \lambda \sum_{n,i,j} \left|w^n_{i,j}\right|^2 \tag{4.10}$$

As deeply analysed in the next section, in a neural context the iterative application of stochastic gradient descent algorithm (SGD) results in the well known weights' update rule

$$w^n_{i,j}(t) = w^n_{i,j}(t-1) - \eta \frac{\partial L(\bar{w})}{\partial w^n_{i,j}} - 2\lambda w^n_{i,j}, \tag{4.11}$$

where η is the learning rate and λ is the regularization parameter.

An analogous procedure for regularization *L*1 yields the update rule:

$$w^n_{i,j}(t) = w^n_{i,j} - \eta \frac{\partial L(\bar{w})}{\partial w^n_{i,j}} - \lambda \tag{4.12}$$

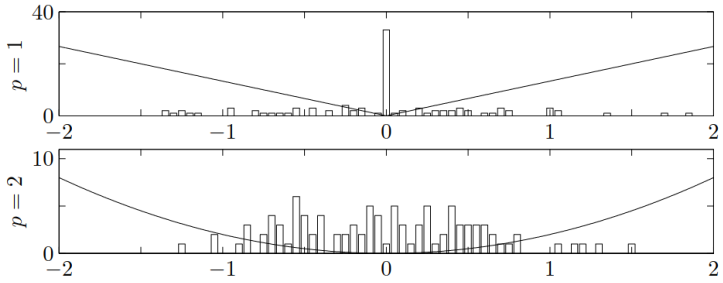


FIGURE 4.1: The number of weights, after training, with the value indicated by the x-coordinate. $L1$ regularization imposes a much more sparse solution on the model, i.e. with many weights set to 0, while the $L2$ regularization is much more permissive. Regularization functions (scaled) are also shown for reference. Image from (Boyd & Vandenberghe, 2004).

In (Han et al., 2015), $L1$ and $L2$ term are tested and used to shrink weights below a user-defined threshold. Those weights are removed and the subnetwork is retrained. The process is carried out several times, until performance decreases.

$L1$ and $L2$ based pruning are widely used: they are easy to implement, do not require a high computational cost and achieve good results in terms of sparsity and performance (Tartaglione et al., 2018).

A disadvantage of these techniques is that the neural weights are driven close to zero without taking into account their relevance in the neural architecture.

A first attempt to deal with this issue is presented in (Tartaglione et al., 2018), where the idea of output-based sensitivity is introduced: weights are selectively penalized depending on their capability to induce variations on network output, when changed.
A limitation of this approach is that it can not be applied at training time, as outputs of the network might not be satisfactory.
A solution to this problem is presented in Lobster (Tartaglione et al., 2022), where the idea of loss-based sensitivity S_L is introduced:

$$S_L\left(L, \mathbf{y}, w_{n,i}\right) = \frac{1}{C}\sum_{k=1}^{C}\left|\frac{\partial L}{\partial y_k}\right| \cdot \left|\frac{\partial y_k}{\partial w_{n,i}}\right| \tag{4.13}$$

Because of the low computational efficiency of (4.13), authors approximate it with the following lower bound

$$S_L \geq \left|\frac{\partial L}{\partial w_{n,i}}\right| \tag{4.14}$$

which allows to define the new update rule:

$$w_{n,i}^{t} = w_{n,i}^{t-1} - \eta\frac{\partial L}{\partial w_{n,i}^{t-1}} - \lambda\Gamma\left(L, w_{n,i}^{t-1}\right)\left[1 - \left|\frac{\partial L}{\partial w_{n,i}^{t-1}}\right|\right] \tag{4.15}$$

λ and η are two positive hyperparameters, and Γ is equal to:

$$\Gamma(y, x) = x \cdot P\left(\frac{\partial y}{\partial x}\right) \tag{4.16}$$

Using an algorithm composed of a regularization phase (in which the described update rule (4.15) is applied) and a pruning phase (where parameters are removed), Lobster yields a significant and competitive reduction of the number of nonzero weights, with a minimal computation overhead.

4.2 The Relevance-Based Pruning Method: Theory

As seen in the previous section, in a neural context, the regularized loss $\tilde{L}$ depends on each weight $w^n_{i,j}$ belonging to layer n and connecting neurons i and j, and has the form:

$$\tilde{L}(\bar{w}) = L(\bar{w}) + \lambda \|\bar{w}\|^2 = L(\bar{w}) + \lambda \sum_{n,i,j} \left| w^n_{i,j} \right|^2 \tag{4.17}$$

The iterative application of stochastic gradient descent algorithm (SGD) at time step t

$$w^n_{i,j}(t) \equiv w^n_{i,j}(t-1) - \eta \frac{\partial \tilde{L}(\bar{w})}{\partial w^n_{i,j}} \tag{4.18}$$

results in the well known weights' decay update rule

$$w^n_{i,j}(t) = w^n_{i,j}(t-1) - \eta \frac{\partial L(\bar{w})}{\partial w^n_{i,j}} - 2\lambda w^n_{i,j} \tag{4.19}$$

where η is the learning rate and λ is the regularization parameter. Therefore the neural weights are driven close to zero without taking into account their relevance in the neural architecture.

In this chapter we present a solution to account for weight relevance proposing a new loss functional $\hat{L}$ modified with respect to eq. (4.17) which leads, via SGD, to a new selective weight decay rule which shrinks the magnitude of only those weights that are not relevant to the final result. For this purpose we modify the regularization term of eq. (4.17) multiplying it by a coefficient which measures how much the final loss value is influenced by modification of $w^n_{i,j}$.

The quantity $\left| \frac{\partial L}{\partial w^n_{i,j}} \right|$ would seem to be a good candidate for this, in fact small derivative values indicate that even a large variation of $w^n_{i,j}$ do not cause large loss variations, i.e. weight changes are not relevant to the final loss value. This derivative anyway is not upper bounded, which is a possible issue in order to preserve SGD convergence properties. Besides, large derivative values are not interesting because we aim at driving to zero (and prune) only irrelevant weights.

Taking account of these requests we therefore define the coefficient of irrelevance $I_{n,i,j}$ as:

$$I_{n,i,j} \equiv \exp\left(-\left|\frac{\partial L}{\partial w^n_{i,j}}\right|\right), \quad 0 < I_{n,i,j} < 1. \tag{4.20}$$

$I_{n,i,j}$ is bounded and assumes values near 1 for irrelevant weights and near 0 for relevant ones. Moreover it has the useful property to be almost everywhere differentiable.

We also define the new loss functional $\hat{L}$, modified with respect to eq. (4.17) to selectively limit the magnitude of weights:

$$\hat{L}(\bar{w}) \equiv L(\bar{w}) + \lambda \sum_{n,i,j}\left(I_{n,i,j} \cdot \left|w^n_{i,j}\right|^2\right) = L(\bar{w}) + \lambda \sum_{n,i,j}\left(\exp\left(-\left|\frac{\partial L}{\partial w^n_{i,j}}\right|\right) \cdot \left|w^n_{i,j}\right|^2\right) \tag{4.21}$$

The iterative application of stochastic gradient descent algorithm to $\hat{L}$ allows to calculate the new weights' update rule:

$$\begin{aligned}
&w^n_{i,j}(t) \equiv w^n_{i,j}(t-1) - \eta \frac{\partial \hat{\mathrm{L}}}{\partial w^n_{i,j}} = \\
&= w^n_{i,j}(t-1) - \eta \frac{\partial \mathrm{L}}{\partial w^n_{i,j}} - 2\eta\lambda \exp\left(-\left|\frac{\partial L}{\partial w^n_{i,j}}\right|\right) w^n_{i,j} - \eta\lambda \left|w^n_{i,j}\right|^2 \cdot \\
&\exp\left(-\left|\frac{\partial L}{\partial w^n_{i,j}}\right|\right)(-1)\frac{\partial}{\partial w^n_{i,j}}\left|\frac{\partial L}{\partial w^n_{i,j}}\right| = \\
&= w^n_{i,j}(t-1) - \eta \frac{\partial \mathrm{L}}{\partial w^n_{i,j}} - 2\eta\lambda \exp\left(-\left|\frac{\partial L}{\partial w^n_{i,j}}\right|\right) w^n_{i,j} + \eta\lambda \left|w^n_{i,j}\right|^2 \cdot \\
&\exp\left(-\left|\frac{\partial L}{\partial w^n_{i,j}}\right|\right) \mathrm{sgn}\left(\left|\frac{\partial^2 L}{\partial {w^n_{i,j}}^2}\right|\right)
\end{aligned} \tag{4.22}$$

As usually done in first order derivative optimization methods, we can neglect the second-order derivative term, so that eq. (4.22) becomes:

$$w^n_{i,j}(t) = w^n_{i,j}(t-1) - \eta \frac{\partial \mathrm{L}}{\partial w^n_{i,j}} - 2\eta\lambda \exp\left(-\left|\frac{\partial L}{\partial w^n_{i,j}}\right|\right) w^n_{i,j} \tag{4.23}$$

Different weights' updates are made in the two cases determined by the extreme values of I:

- $I \sim 0$: in this case the weight is relevant. The third term in eq. (4.23) is roughly zero, so a classic update is performed, without targeted reduction of weight norm.
- $I \sim 1$: in this case the weight is irrelevant. Because $I \sim 1$ implies $\frac{\partial L}{\partial w^n_{i,j}} \sim 0$, the second term in eq. (4.23) is roughly zero, and the update

rule becomes:

$$w^n_{i,j}(t) \simeq w^n_{i,j}(t-1) - 2\eta\lambda w^n_{i,j} \tag{4.24}$$

We can see that in this case the weight is actually driven to zero at each iteration, because if $w^n_{i,j} > 0$ then $\Delta w^n_{i,j} < 0$ (i.e. the norm of a positive irrelevant weight is decreased); on the other hand if $w^n_{i,j} < 0$ then $\Delta w^n_{i,j} > 0$.

4.3 The Relevance-Based Pruning Method: the Algorithm

The pruning algorithm proceeds this way:

1. We get a state-of-art checkpoint, either founding it in literature or training it on our own.
2. We finetune the checkpoint by using our proposed regularization term, as in equation (4.21). In this step any optimizer can be used. During finetuning we regularly evaluate the model performances on the validation set every *evaluation-interval* steps, and:
   - **prune**. If the validation performance is higher than a user-defined *lower-bound*, a fixed percentage (*pruning-percentage*) of the remaining model parameters is pruned, choosing from the ones with lower magnitude.
   - **not prune**. If the validation performance is lower than the user-defined lower-bound, the model is not ready to be pruned, so the finetuning proceeds normally.
3. These last two steps of the finetuning process are iteratively repeated until the model reaches a validation performance plateau (so it can not be pruned further).
4. We perform a final finetuning phase without regularization, aiming to get the best performing checkpoint.

For this process is necessary to define some hyperparameters:

- *evaluation-interval*: number of steps between two validation performance assessments.
- *lower-bound*: performance lower-bound. For the Image Classification task the performance is measured by accuracy, while for the Language Generation task we use BLEU Papineni et al., 2002. The lower-bound is chosen as slightly lower than the state-of-the-art performance.
- *pruning-percentage*: The percentage of remaining non-zero parameters to be pruned at every pruning step.

If not stated otherwise, training and finetuning hyperparameters are chosen via grid-search.

When advanced in training, usually the model reaches an accuracy plateau making difficult for the algorithm to hit a pruning step. Because of this we introduce an exponential decay schedule on λ which decreases the regularization term between two validation step. This procedure favours accuracy with

respect to sparsification and helps the model to break the performance plateau and to cross over the lower bound leading to further pruning.

4.4 Experiments

The first studies concerning sparsity in language architectures appeared recently (Gale et al., 2019; Molchanov et al., 2017), following the progressive replacement of recurrent architectures by transformer-based models, and mainly focused on attention heads pruning (Michel et al., 2019; Voita et al., 2019). Nonetheless, with respect to the great amount of available sparsity techniques for imaging, it is yet rare to find as many results in the context of language generation.

We tried to fill this gap sparsifying a Transformer architecture in the framework of two different language tasks: Dialog Learning and Machine Translation. Since our sparsification algorithm is not architecture-specific, no modifications are needed with respect to what described in Sec. 4.3.

We implemented the model using the Huggingface[1] library which provides easy access to different datasets, tokenizers and output generation techniques. We made all our experiments using one Nvidia TITAN RTX 24Gb GPU.

4.4.1 Datasets

WMT14 (Bojar et al., 2014) is a collection of datasets presented in the Ninth Workshop on Statistical Machine Translation. It is made by parallel corpus i.e. dataset with the same sentences translated in different languages. It is derived from many different sources, among which there are the Europarl corpus Koehn, 2005 (created from the European Parliament Proceedings in the official languages of the EU), the News Commentary Tiedemann, 2012 corpus and the Common Crawl corpus CommonCrawl, 2012 (which was collected from web sources).

For our experiments we use the English to German translation dataset En-De WMT14, provided by the Stanford Natural Language Processing Group [2], which is more than 300x bigger than the Taskmaster-1; it contains 4468840 training samples and 3000 test samples. Some source-translation examples is shown in Table 4.1.

TABLE 4.1: Example of translation pairs from En-De WMT14.

Source **Translation**
Iron cement protects the ingot against the hot, abrasive steel casting process. Nach der Aushärtung schützt iron cement die Kokille gegen den heissen, abrasiven Stahlguss.
Goods and services advancement through the P.O.Box system is NOT ALLOWED. der Vertrieb Ihrer Waren und Dienstleistungen durch das Postfach System WIRD NICHT ZUGELASSEN.
Their achievement remains one of the greatest in recent history. Das bleibt eine der größten Errungenschaften in der jüngeren Geschichte.

Taskmaster-1 Already described in Section 3.3.1.

[1] https://huggingface.co/
[2] https://nlp.stanford.edu/

TABLE 4.2: Transformer hyperparameters.

Hyperparameters	Taskmaster-1	WMT14
vocabulary	32k	32k
# encoder layers	2	6
# decoder layers	2	6
# attention heads	4	8
feed forward dim	256	2048
embedding dim.	256	512
# weights	10M	61M
max sequence len.	256	256
beam size	6	4
length penalty	-	0.6

4.4.2 Transformer on WMT14

Transformer architecture details are described in Table 4.2 and follow the settings from Gale et al., 2019.

In order to obtain the initial checkpoint, we train the model for 10 epochs with batch size = 120, $\eta = 5e-05$, using Adam optimizer ($\beta_1 = 0.85, \beta_2 = 0.997, eps = 1e-8$) and reach comparable BLEU performance with the baseline defined in Gale et al., 2019.

Starting from the checkpoint described above, the finetuning with regularization phase continues for 16 epochs with batch size = 100, $\eta = 2.5e-05$ and $\lambda = 2.22e-07$. Evaluations on the validation set is done every 6000 steps (24 times each epoch) with BLEU lower-bound = 27.3 and pruning 1% of the remaining weights when required. Finally we finetune without regularization for 5 more epochs. With respect to Gale et al., 2019, we stop pruning when the validation performance reaches a plateau (or drop) and never re-climb the user-defined lower-bound, as described in Section 4.3. This criterion causes the pruning to stop at $\sim$80 % of sparsity.

As can be seen from Figure 4.2, our pruning technique performs better than all other methods, preserving BLEU values up to $\sim$75% of sparsity while dropping at most 0.5 with respect to the baseline. It also seems to be more resilient at higher compression levels since BLEU scores start to degrade visibly only after $\sim$75 % of sparsity is reached, whereas the other five pruning methods degrade earlier.

TABLE 4.3: Layer-by-layer and global pruning results on WMT14.

Model	Residual Weights (%)					BLEU	Sparsity (%)
	Encoder Attention	Decoder Attention	Encoder FFN	Decoder FFN	Embedding/ Classification Head		
Baseline	100	100	100	100	100	27.20	–
Our model	24.70	27.54	21.27	20.10	12.43	25.56	79.93

Table 4.3 shows in detail layer-by-layer and global pruning percentages at the higher compression level reached.

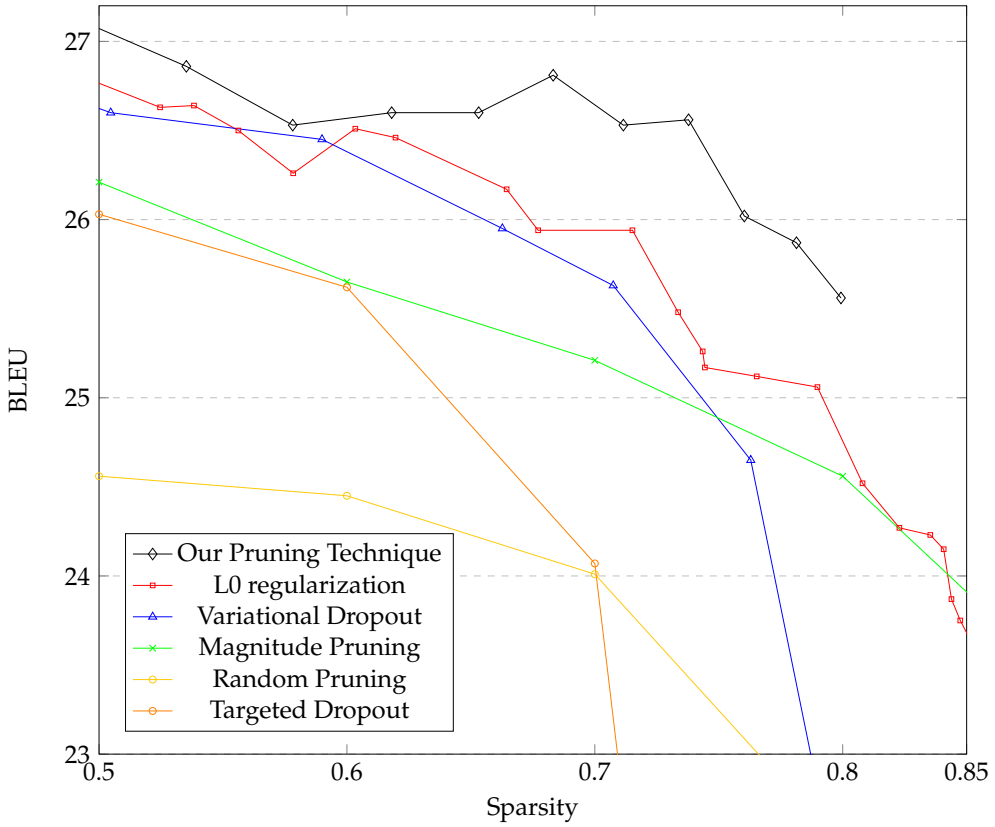


FIGURE 4.2: BLEU results comparison at different sparsity levels on WMT14 dataset. Except for our technique, the datapoints are taken from Gomez et al. (2019) for targeted dropout and from Gale et al. (2019) for the others methods, for which we take only their best runs (in terms of BLEU).

4.4.3 Transformer on Taskmaster-1

Following Byrne et al., 2019 we use as input-data the dialog context up to the last user turn, and as target the subsequent assistant utterance. An example of this format is shown in Table 3.3.

Our Transformer has the same architecture details presented in Byrne et al., 2019, and shown in Table 4.2. We train it for 15 epochs using the Adam optimizer ($\beta_1 = 0.85, \beta_2 = 0.997, eps = 1e-8$) with batch size = 150 and dropout = 0.2. The final checkpoint we obtain shows comparable BLEU performance to the author's one.

Starting from this checkpoint, we finetune with regularization for 40 epochs with $\eta = 0.0005$. We rely on a small $\lambda = 1e-05$ not to lose performance during the early stages. We find that checking every 300 training steps (i.e. 4 times every epoch) is a good compromise to get frequent pruning steps while retaining generation ability. The BLEU lower-bound is set to 6.03, which is very close to the author's baseline result of 6.11, and we prune with pruning-percentage = 0.1. After 30 epochs, the algorithm makes the last pruning, and the last 10 epochs are used to recover BLEU score.

As discussed above, to the best of our knowledge there are no achievements

TABLE 4.4: Layer-by-layer and global pruning outcomes on Taskmaster-1.

Model	Residual Weights (%)					BLEU	Sparsity (%)
	Encoder Attention	Decoder Attention	Encoder FFN	Decoder FFN	Embedding/ Classification Head		
Baseline	100	100	100	100	100	6.11	–
L1	2.53	18.57	1.17	53.08	20.99	6.15	79.49
L2	17.71	21.20	15.21	26.61	6.28	6.15	**90.10**
Our model	21.53	21.77	21.68	26.93	7.12	**6.21**	90.09

yet in literature about weights sparsity in dialog generation tasks. We therefore establish in this context the first results, testing our method along with two baselines (regularization L1 and L2), resumed in Table 4.4.

Our sparsification technique allows to obtain a highly sparsified model, with a sparsity level greater than 90%. Moreover, our final BLEU is higher than the original result, suggesting that in some cases a sparsified model is able to generalize better then a non-sparsified one.

Chapter 5

Conclusions and Future Work

This thesis discusses about some important features and properties a neural language generation system should have in order to output faithful and informative text. We propose that a reliable model should have the ability to copy information from the input, to be more precise and specific, and to avoid generating dull or not informative text by relying more on previous experience, like the one coded in the training set. Both these ideas are explored: in Chapter 2 we presented an end-to-end generative model paired with a character based copy mechanism. This system successfully deals with the rare word problem, and it is further improved by a specific training procedure which squeezes data more efficiently, exploiting the double-way alignment between input and output. In Chapter 3 the focus is on augmenting the generative abilities of a Transformer-based model by retrieving precise information during inference and leveraging the training data to disambiguate possible next turn continuations in the dialog learning framework. This solution makes the system less prone to output general inconclusive sentences like "Thank you" or "I don't know" as confirmed by a human evaluation experiment (detailed in Appendix A). Overall the techniques presented in this thesis thrive to use the available data at its maximum; being it exploiting some biases like using the data time period to finetune the model or choosing more suitable retrieval datastores, or by leveraging fine grained information via token-level retrieval or via a copying mechanism.

On the other hand deep learning models show the tendency to require more and more layers and parameters, leading to difficulties in integrating them in normal hardware, or more simply heading towards overfitting or over-parametrization issues. In Chapter 4 we argue that "big models" are not a strict requirement for good or even state of the art results and show how our sparsification-pruning techinique can shrink generative language models down to one tenth of their original weight number, still resulting in well trained and effective deep learning networks. Moreover,t our algorithm is very general and can be used for other architectures and tasks as well, like image classification with convolutional deep learning models (see Appendix B).

Still our solutions have drawbacks that we plan to tackle in future works. For example, the overhead of integrating a retrieval system with a language model is not negligible both in terms of memory and computational-time. One solution could be to use a sentence-level retrieval system instead of our token-level index; this could lead to smaller datastores and faster inference since the

search time would be lowered significantly. Nonetheless, this will practically lead to a ranker model, losing the generation ability of our solution, which we argue being important for real general dialog models. On the sparsificaiton side, it would be interesting to test our algorithm on specific hardware architectures which expose routines made ad hoc for efficient operations on sparsified matrix, improving both training and inference speed. We hope that the advent of specialized hardware in future years will make this possibility easily implementable.

Appendix A

Deepening Transformer-XL + kNN Performance on Dialog Generation

A.1 Playing with Different Data

For this experimentation we use two different proprietary datasets called D1 and D2, which are all collected from real customer care agent-user dialogs over a 1 year period. Test and development sets are built from the more recently collected chats in the datasets. We want to assess models' performance on data which show the most similar distribution to data it will face at deployment time (production). Focusing on modeling data too old in the past would lead to degrading performance since the model would learn biases that may not apply anymore. For example some call center agents may have left and other may have joined, some products that were available in the past may not be present anymore, or agent may be instructed to answer differently to specific user questions. For this reason we explore different indexes (see Sec. 3.3.4) and finetuning sets always collected from the end of the year-period going backward (not overlapping with test and development sets).

The model used is a Transformer-XL as described in 3.4.2 and all the experiments presents meta-information (i.e. turn-number and agent-id) within chat texts. Moreover, all the names in the datasets were swapped with a special `PERSON` token beforehand.

Information about the two datasets used for our experiments are shown in Tab. A.1. The indexes, derived from sub-sets of the training data are collected over different time periods (i.e. 3-months, 1-month and 2-weeks) differing in their number of embeddings and the amount of disk space they use, as shown in Tab. A.2. Figures A.1 and A.2 present our model results when it is used paired with different indexes (labeled as described above) and different λ values. Both figures show that the Transformer-XL + kNN greatly improve BLEU

TABLE A.1: Details of D1 and D2 datasets.

name	train chats	dev. chats	test chats
D1	2M	6k	6k
D2	1.3M	6k	6k

TABLE A.2: number of contexts' embeddings in the datastore and dimension of the relative indexes on disk.

index		**# embeddings**	**dimension on disk**
D1	3 months	82M	3.3GB
	1 month	26M	1.1GB
	2 weeks	12.5M	520MB
D2	3 months	90M	3.9GB
	1 month	30M	1.3GB
	2 weeks	15M	620MB

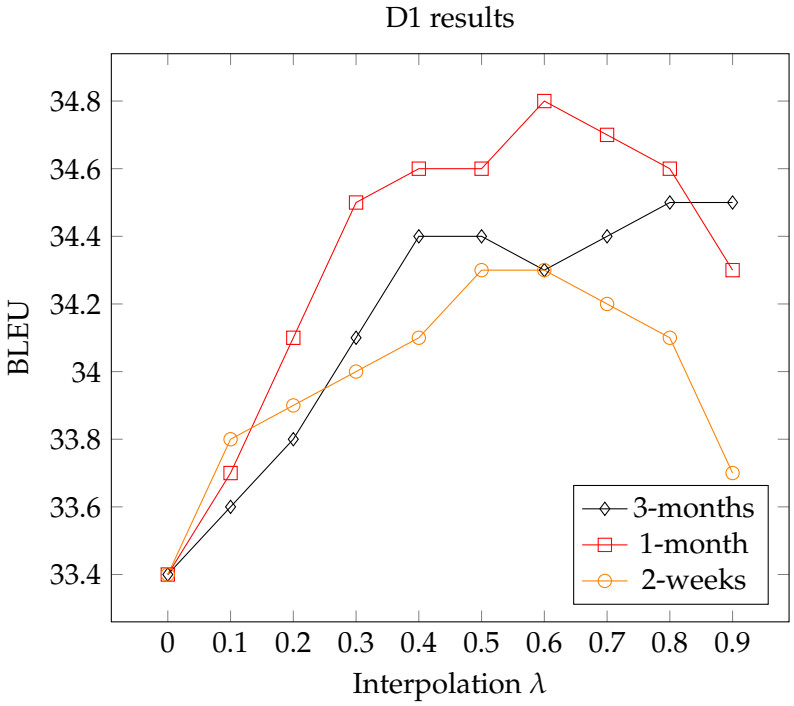


FIGURE A.1: BLEU test results comparison at different interpolation λ levels on D1 dataset.

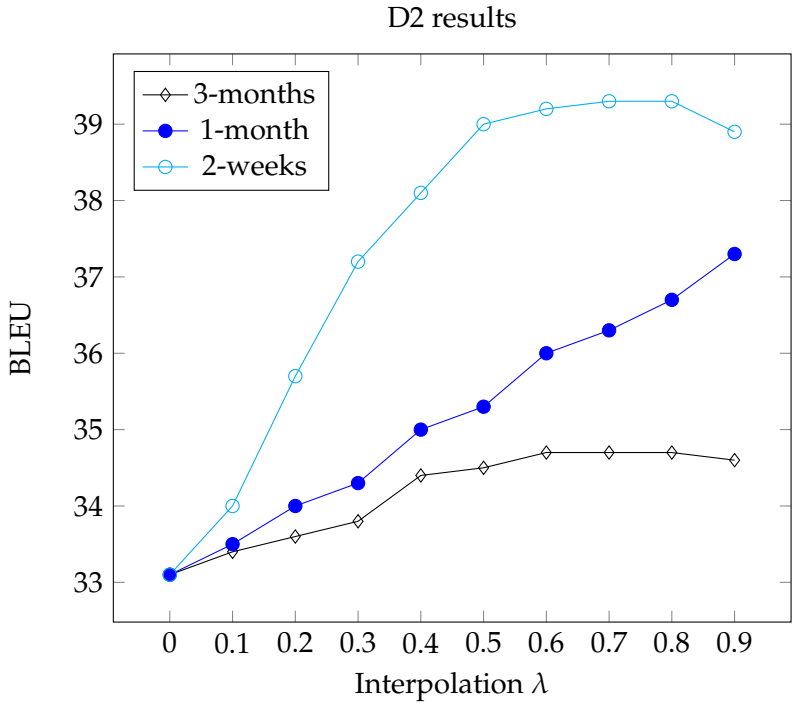


FIGURE A.2: BLEU test results comparison at different interpolation λ levels on D2 dataset.

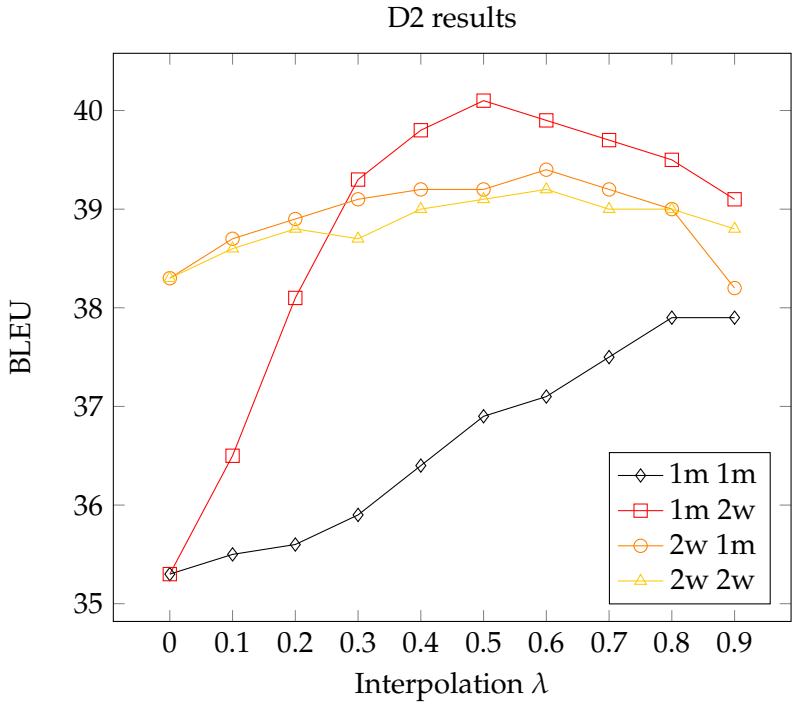


FIGURE A.3: BLEU test results comparison at different interpolation λ levels on D2 dataset. 1m and 2w in the legend mean 1-month and 2-weeks respectively and, as an example, the label "1m 2w" means that the model has been finetuned on 1-month data before using it to create the 2-weeks index.

performance with respect to the Transformer-XL baselines ($\lambda = 0$ in the figures). For D1 dataset the best interpolation point is at $\lambda = 0.6$ when using the 1-month index with a gain of 4.2% over the baseline, and for D2 the best interpolation points are at $\lambda = 0.7$ and $\lambda = 0.8$ when using the 2-weeks index, with a BLEU gain of 15.7%. As a further step, we study what happens when our model is finetuned on the same indexes' data before the index creation. So, in other words, the trained Transformer-XL is finetuned on subsets of the training data itself and then it is used to encode these datastores. So doing we suppose that the model could have a better ability to encode the actual indexes's data since it has been overtrained on them. Subsequently index search precision should improve and lead to better generation performance. Figure A.3 shows BLEU results when the finetuning and index data are the last 1-month or 2-weeks of the training set. Comparing with the baseline BLEU=33.1 shown in Fig. A.2, finetuning gives some improvements even without kNN interpolation, reaching BLEU=35.3 and BLEU=38.3 respectively for 1-month and 2-weeks. Looking at different interpolation levels it seems that using the 2-weeks data is a good choice no metter if they are used as datastore or for finetuning, but the best combination is finetuning for 1-month and indexing 2-weeks which lead to BLEU=40.1.

A.2 Human Evaluation

In order to better understand the quality of the Transformer-XL + kNN generation we performed a qualitative analysis via a human evaluation. We used two datasets: the D2 dataset (already presented in A.1) and the D3 dataset (see Table A.3). Similarly to previus datasets, D3 is a proprietary collection of data which contains agent-users chats gathered from a real customer center service. The peculiarity of this dataset is that it comes with two different test sets: the *beta* test set which contains all the possible agent-ids as usual, and the *alpha* test set which contains the top-84 agents according to an internal metric from the customer service.

TABLE A.3: Details of the D3 dataset.

train chats	**dev. chats**	**test chats**	**# embeddings**	**dim. on disk**
54k	6k	6k (alpha) 6k (beta)	6M (1-month index)	260MB

The human evaluation is carried out comparing outputs from the Transformer-XL + kNN and from a proprietary ranking model which is described in our patent (D. Liu et al., 2021). This latter model, called BLSTM + KD-tree, is based on a Bi-LSTM encoder architecture which encodes contexts and entire turn continuations while learning to minimize the cosine distance between related embeddings. During inference, a trained ranker scores a set of turn continuations which are chosen from the specific agent turns and are organized by a KD-tree structure.

Since the proprietary system is a ranker model, while the Transformer-XL + kNN is a generative one, we decided to compare the top-5 output candidates from the ranker with the top-5 beam generated candidates from our model, simply scored by the probabilities given by the beam search, in order to have

a more complete evaluation. We collected the outputs from our system and the BLSTM + KD-tree for 300 samples (randomly chosen) from each of the 3 different test sets, two from D3 and one from D2. These 9000 utterances (i.e. 300 samples * 3 test sets * 10 systems outputs) are organized and presented to human evaluators to annotate as described in the next Section A.2.1.

A.2.1 Data Preparation

Data is divided in 9 *chunks* and every human evaluator gets to annotate 1 of them. Each chunk is composed of 100 *blocks* and one block is composed by a test sample with its own systems' continuations. A block example is shown in Figure A.4 and presents the following information:

- *Header*: it contains the question-id and a counter which are information needed to easily gather the results once the annotation is done.
- *Agent alias*: it is the name of the agent talking to the user.
- *Context*: it contains a sequence of agent-user turns. This is the input context given to the two models.
- *Reference*: it is the real next agent turn following from the context (i.e. the target).
- *First stage*: it usually contains 2 candidate agent turns and their indexes (i.e. 0 and 1). This utterances are the top-1 output from the Transformer-XL + KNN and the BLSTM + KD-tree. The annotators need to choose which one is the best.
- *Second stage*: contains a variable length list of candidate agent turns with their indexes starting from the last index in first stage. These turns are the other candidates but the ones in first stage. Here the annotators are asked to point which of them are acceptable.

In order to make the annotation process more slim and quick for the evaluators, if two or more candidates are equal just one of them is presented in the block. If the two top-1 candidates are equal (i.e. both system exactly agree on the next agent turn) the entire first stage is dropped, since showing just one of them would make the question "which one is the best?" meaningless. Instead the top-1 candidate is added to the second stage to be evaluated as ok, or not ok, among the others. Moreover, the candidates are not presented in a predefined order since this could lead to the rise of unwanted annotator's biases. For example, if the first top-1 turn in first stage is always the output of a specific system, and this usually is preferable against the second top-1, then the annotator could lead more easily toward choosing the first turn as best when in doubt.

A.2.2 Annotator's Instructions

Human evaluators[1] are 9 employees from the same company the proprietary datasets come from. They are asked to follow some instruction to annotate the chunk files assigned to them.

1. Read the `Context`.

[1] "evaluators" is used as a synonym of "annotators".

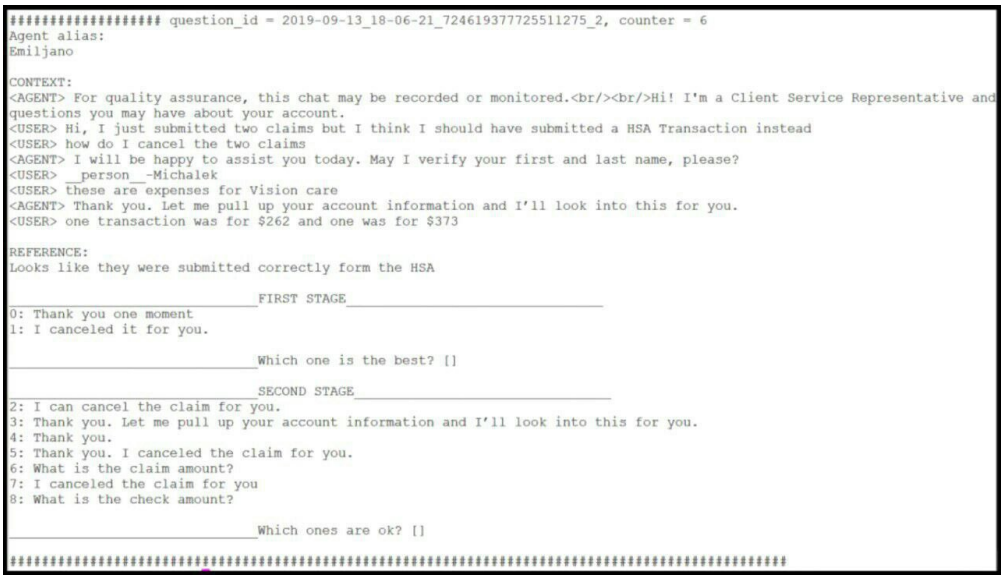


FIGURE A.4: A block example, ready to be annotated.

2. Read the `Reference`.
3. Read the 2 sequences in `first stage` (if it is present) and choose which one is the best as a possible continuation for the context.
    - If you can not decide because the two candidates share the same meaning, or they are equally bad, just leave the "[]" after "Which one is the best?" as it is.
    - Otherwise write the chosen sentence's index within the square brackets. Example of the syntax "Which one is the best? [1]".
4. Read the sequences in `Second stage` and for each one evaluate if it is acceptable as context continuation or not.
    - This evaluation must be done also for the two sentences in the first stage (if it is present).
    - Write the chosen sentences' indexes next to "Which ones are ok?" between the square brackets. Example of the syntax "Which ones are ok? [0 1 4 5]".

Other info given to the annotators:

- Please, try to give a preference in the `first stage`.
- Weird punctuation formatting like "it ' s" or the presence of tags like "`__PERSON__`" in text should not be taken into consideration for the final judgment.
- When evaluating a sentence, compare it to the reference to check if the reference information are shown. If the sentence shows more information then the reference, use your own knowledge of the world to assess if it is sensible or not.
    - Example: candidate -> "i don't know", even though it is a grammatically correct utterance and can fit virtually any context, it is not an

acceptable next turn if the reference contains more interesting information like: reference -> "I don't know, but could check and verify your amount at this URL:...".

– Example: candidate -> "There is no minimum balance required to open or maintain the account", reference -> "There is no minimum investment required to open the account". In this case we do not know if there is a cost in maintaining the account since the reference does not say it. So using our knowledge we could still evaluate it as ok.

A.2.3 Results

The results obtained by the Transformer-XL + kNN and the BLSTM + KD-tree are shown in Table A.4 for the BLEU and Exact Match metric. Exact Match metric it is just the percentage of times the model output is equal to the reference turn after both are lowercased and normalized for punctuations tokens.

The results about the evaluation of the first stage are shown in Table A.5 where there is the total amount of times each system's top-1 candidate has been chosen as best. The results show high agreement between the automatic metrics and the human evaluation giving the Transformer-XL + kNN as superior with respect to the BLSTM + KD-tree. In the first stage our model is evaluated as best up to 66% more times then the BLSTM + KD-tree (i.e. D3 alpha case). However, more than half of the times the evaluators preferred not to take a decision, considering the two systems very similar.

The second stage, in Table A.5, shows that even though our model seems to be superior by human annotators, the BLSTM + KD-tree is not giving much worse sentences. In fact, the difference between the number of times our model has generated acceptable sentences vs. the BLSTM + KD-tree is 10.7% on average, among the 3 test sets evaluated utterances. For the Transformer-XL + kNN case, on average choosing the top-1 sentence, without considering the other candidates, is increasing the probability to miss a good continuation by 57,6% with respect to considering all the possible continuations among its top-5. For the ranker model the increase in missed chances is 56,8%. This suggests that the retrieved utterances are as different among them as the beam generated ones.

TABLE A.4: BLEU and Exact Match metrics for our model and the BLSTM + kD-tree.

Dataset	BLEU		Exact Match	
	BLSTM + KD-tree	**Transformer-XL + kNN**	**BLSTM + KD-tree**	**Transformer-XL + kNN**
D2	35.6	40.1	28.8	32.7
D3 beta	24.8	26.2	20.2	21.0
D3 alpha	24.6	30.4	20.4	24.4

A.3 Template Constrained Generation

It is not guaranteed that a generative model, like the Transformer-XL (+kNN), actually outputs sensible sentences every time. Hallucinations are surely an issue, but they are not the only problem that a generated output may present.

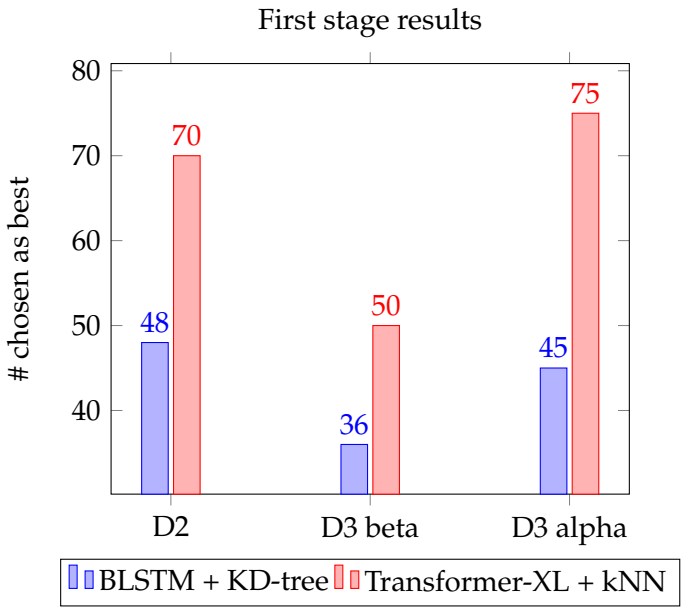


FIGURE A.5: Number of times the top-1 solution has been chosen as best. The maximum value would be 300 if the given model is best 100% of the times.

TABLE A.5: Cumulative number of times the top-1, top-3 and top-5 solutions have been evaluated acceptable as next turns. The maximum value would be 300*x if the given model top-x candidates are always acceptable next turn.

Second Stage	model	D2	D3 beta	D3 alpha
Top-1	**BLSTM + KD-tree**	96	109	148
	Transformer-XL + kNN	103	121	165
Top-3	**BLSTM + KD-tree**	147	149	196
	Transformer-XL + kNN	165	168	224
Top-5	**BLSTM + KD-tree**	166	166	215
	Transformer-XL + kNN	184	178	236

TABLE A.6: D4 dataset dimensions.

train chats	**dev. chats**	**test chats**	**# embeddings**	**dim. on disk**
100k	6k	6k	12M (1-month index)	550MB

For example, it is possible to - output word loops, where the system is repeating a certain n-gram over and over; - output not entailing sentences where one piece of text does not follows from the hypothesis; - output non grammatically correct sentences or mixed words if using vocabulary which allows for that. One simple solution we tried that alleviates all this problems is to force the model to generate n-grams already seen in training. This allows to get a trade off between the generalization capability of the model and its reliability. We call this method *template constrained generation* since the utterances are conforming to a predefined set of possibilities, similar to templates.

We test this idea by gathering all the possible 100-grams from the training set and then, at inference time, forcing the Transformer-XL + kNN to discard answers which do not conform to them. In order to check if a sentence is actually a valid n-gram we perform a check at every model generation step; the probability to generate a specific token is set to 0 if this would generate an invalid n-gram. The datasets used are the D2 dataset (already presented in A.1) and the D4 dataset (see Table A.6), which is another proprietary dataset similar to the ones already presented. It is a collection of data which contains agent-users chats gathered from a real customer center service, it comes with meta-information for every turn and has been pre-processed to swap names with `__PERSON__` tags.

Figure A.6 presents the BLEU results for the template constrained generation. A minimal drop in BLEU is evident when using the template generation on the D2 dataset. On the D4 dataset instead, we see that for low interpolation ($\lambda \leq 0.3$) the model have a minimal BLEU gain. More interestingly, for higher λ the template generation seems to force good quality outputs while the standard generation would drop performance for overinterpolation. Overall it is difficult to assess if the template generation is leading to significant change in BLEU metric with respect to the standard one, but surely it gives more guarantees about the generated output leading to more reliable results.

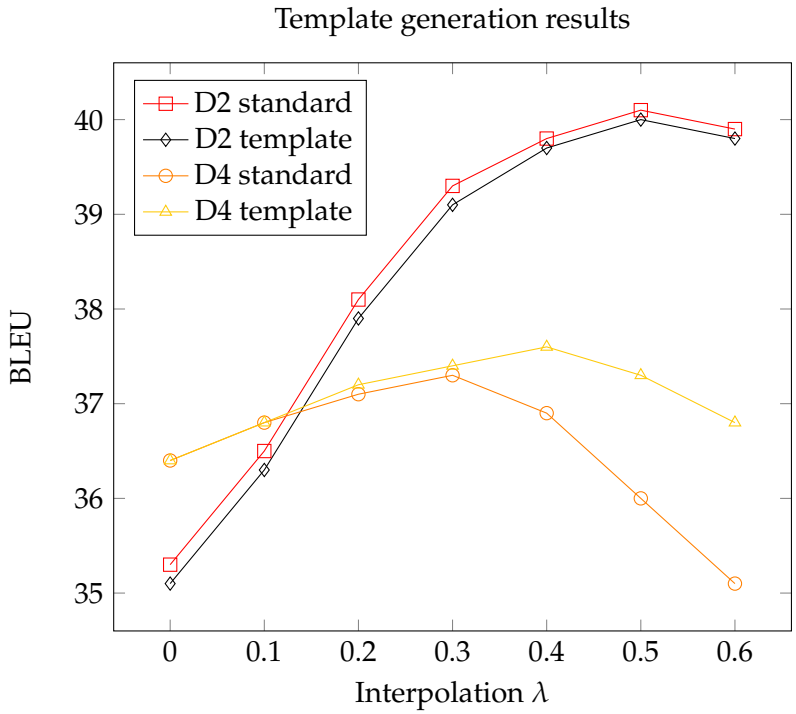


FIGURE A.6: BLEU test results on the D2 and D4 datasets for the template generation vs. the standard generation. Results are shown at different interpolation λ levels.

Appendix B

Sparsity on Imaging

B.1 Imaging

We test our sparsity method on four different Image Classification datasets chosen among the most popular benchmarks in literature for sparsity research.

B.1.1 Datasets

MNIST LeCun and Cortes (1990) is composed of 70,000 grey-scale images containing handwritten numbers, whose dimensions are 28x28; the database is split into training set (60,000 images) and test set (10,000 images).

Fashion-MNIST Xiao et al. (2017), based on the assortment of Zalando's website, is a dataset comprising of 28x28 grayscale images of 70,000 fashion products from 10 categories, with 7,000 images per category. Like MNIST, the training set has 60,000 samples and the test set 10,000. Fashion-MNIST, although similar to MNIST, is more challenging.

CIFAR-10 (from Canadian Institute for Advanced Research) Alex Krizhevsky and Hinton (2009) is a subset of the Tiny Images (Torralba et al., 2008) dataset and consists of 60,000 32x32 color images labelled with one of 10 mutually exclusive classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. There are 6000 samples per class, split in 5000 for training and 1000 for testing.

ImageNet Deng et al. (2009) Arranged according to the WordNet (Miller, 1995) noun hierarchy, ImageNet is a real image database, in which each node of the hierarchy is represented by thousands of images; it contains more than 20,000 categories. In total, 14 million pictures were hand-annotated and for one million of those the bounding boxes were also provided. The RGB images have an average size of 469x387 pixels, but are usually preprocessed by sampling them at 256x256 pixels.

The above described datasets have been processed using the neural architectures which produce current state-of-art results; they are detailed in the following.

TABLE B.1: Hyperparameters used in imaging experiments.

Hyperparameters	LeNet-5	ResNet32	VGG16
# epochs	120	290	30
# batch size	100	128	100
η	0.001	0.0005	0.0001
Optimizer	Adam	SGD	SGD
lower-bound	98.7	92.9	68.5
λ	0.001	1e-6	1e-4
pruning-percentage	4 %	4 %	1 %
eval-interval	250	25	2500

B.1.2 LeNet-5 on MNIST

LeNet-5 (LeCun et al., 1989) consists of a Convolution (Conv) layer with 6 5x5 filters, a 2x2 Pooling layer, a Convolution layer with 10 5x5 filters, another 2x2 Pooling Layer and three fully connected (FC) layers (120, 84, 10), for a total of 431080 parameters.

Since MNIST is an easy dataset, a pretrained checkpoint is not necessary, so we directly trained with regularization from scratch, using hyperparameter values resumed in table B.1. Finally we finetuned without regularization for 5 additional epochs. Results from our model and competitors are shown in table B.2, together with the performance of the not sparsified baseline model.

TABLE B.2: Test results for LeNet-5 architecture on MNIST dataset.

	Residual Weights (%)				Accuracy (%)	Sparsity (%)
Methods	Conv1	Conv2	FC1	FC2		
Baseline	100	100	100	100	99.32	–
Han et al., 2015	66	12	8	19	99.23	91.59
Tartaglione et al., 2018	67.6	11.8	0.9	31.0	99.22	98.04
Y. Guo et al., 2016	14	3	0.7	4	99.09	99.09
Ullrich et al., 2017	-	-	-	-	99.03	99.38
Tartaglione et al., 2022	22	2.38	0.22	5.98	99.21	99.56
Louizos et al., 2018	45	36	0.4	5	99.00	98.57
Molchanov et al., 2017	33	2	0.2	5	**99.25**	99.64
Our method	29	1.82	0.11	3.35	99.23	**99.71**

Performances on the MNIST dataset are very similar to each other since accuracy and sparsification on this simple task have reached the top possible ones, i.e. very close to 100 %. With our method we obtain less than 1,5k non zero residual weights, a result primarily due to a better sparsification of the fully connected layers, where the majority of the weights are. We obtain almost the best accuracy with the only exception of Sparse VD even though with higher sparsity. We also note that we get the same result of Han et al. (2015) but with 8% less weights.

B.1.3 LeNet-5 on Fashion-MNIST

We obtain the initial checkpoint after training for 21 epochs. We then use the same hyperparameters shown in table B.1 for finetuning with regularization, except for lower-bound = 90.5 and epochs = 75; finally we finetune without regularization for 50 additional epochs. Table B.3 compares performances on this dataset.

As we can see our method reaches the best performance both in terms of accuracy and compression, which is obtained thanks to high sparsification of the fully connected layers.

TABLE B.3: LeNet-5 on Fashion-MNIST.

	Residual Weights (%)				Accuracy (%)	Sparsity (%)
Methods	Conv1	Conv2	FC1	FC2		
Baseline	100	100	100	100	91.90	–
Tartaglione et al., 2018	76.2	32.56	6.5	44.02	91.50	91.48
Han et al., 2015	-	-	-	-	91.56	93.04
Tartaglione et al., 2022	78.6	26.13	2.88	32.66	91.53	95.70
Our method	78.84	17.84	1.20	6.26	**91.70**	**97.66**

B.1.4 ResNet32 on CIFAR-10

ResNet32 (K. He et al., 2016) is composed of a Convolution layer with 16 3x3 filters, a batch normalization layer, 3 ResNet layers and a final dense layer, for a total of 464154 trainable parameters. All the ResNet layers are composed of 5 ResNet blocks, with different configuration: they all share 2 Batch Normalization layers, but differ in the number of kernels generated by the 2 Convolutional layers (16 3x3 filters for the blocks of the first ResNet layer, 32 3x3 for the second ResNet layer, 64 3x3 for the second ResNet layer).

We finetune with regularization using the hyperparameters shown in table B.1, and subsequently finetune without regularization for 1 last epoch with a batch size = 64 and η = 6e-5. The initial checkpoint is obtained with the same hyperparameters, but training for 200 epochs with $\eta = 0.1$.

Table B.4 show that our technique on this more challenging task, which involves a deeper neural architecture and real images, outperforms in terms of sparsity all competitors. With respect to accuracy, both our method and Tartaglione et al. reach the the baseline values.

B.1.5 VGG-16 on ImageNet

VGG-16 is composed by 13 Convolutional layers with 3x3 filters followed by Relu activations. These layers are interleaved by 5 MaxPooling layers. Finally an AdaptiveAveragePool layer with 7x7 dimension is used before a sequence of 3 Linear layers with Relu activations which lead to the 1000-dimensional output. The model consists of 138357544 parameters and for our experiment we start our regularized finetuning from a pretrained checkpoint[1] using the

[1]https://pytorch.org/vision/stable/index.html

TABLE B.4: ResNet-32 on CIFAR-10.

Methods	Accuracy (%)	Sparsity (%)
Baseline	92.67	–
Molchanov et al., 2017	92.12	50.11
Louizos et al., 2018	91.20	60.00
Han et al., 2015	91.92	71.51
Gomez et al., 2019	92.54	80.00
Tartaglione et al., 2022	**92.67**	80.11
Our method	**92.67**	**81.27**

hyperparameters shown in table B.1. We further finetune for other 30 epochs without regularization to get the best accuracy.

TABLE B.5: VGG-16 on ImageNet.

Method	Residual Weights (%) Convolutional	FC	Accuracy (%)	Sparsity (%)
Baseline	100	100	71.30	–
Han et al., 2015	32.77	4.61	68.66	92.49
Tartaglione et al., 2018	56.49	2.56	69.08	**92.91**
Our model	37.48	3.57	**69.48**	92.82

Results show that VGG-16 is highly overparameterized for the task, since all methods are able to prune almost 93% of the network's parameters, shrinking them from $\sim$138M to less than 10M. Our method confirms its ability to scale to big model ($\sim$100M weights) obtaining the best accuracy among the shown methods.

Bibliography

Agarwal, S., & Dymetman, M. (2017). A surprisingly effective out-of-the-box char2char model on the E2E NLG challenge dataset (K. Jokinen, M. Stede, D. DeVault, & A. Louis, Eds.). *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, Saarbrücken, Germany, August 15-17, 2017*, 158–163. https://doi.org/10.18653/v1/w17-5519

Aharoni, R., Goldberg, Y., & Belinkov, Y. (2016). Improving sequence to sequence learning for morphological inflection generation: The BIU-MIT systems for the SIGMORPHON 2016 shared task for morphological reinflection. *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, 41–48. https://doi.org/10.18653/v1/W16-2007

Alex Krizhevsky, V. N., & Hinton, G. (2009). CIFAR RGB image dataset. https://www.cs.toronto.edu/~kriz/cifar.html

Al-Rfou, R., Choe, D., Constant, N., Guo, M., & Jones, L. (2019). Character-level language modeling with deeper self-attention. *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, 3159–3166. https://doi.org/10.1609/aaai.v33i01.33013159

Apple. (2021). Apple siri [The Apple Inc. virtual assistant (version macOS Monterey 12.0.1)]. www.apple.com/it/ios/siri/

Ba, L. J., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *CoRR*, *abs/1607.06450*. http://arxiv.org/abs/1607.06450

Badeva, V., & Morozov, V. (1991). *Problèmes incorrectement posés: Théorie et applications en identification , filtrage optimal, contrôle optimal, analyse et synthèse de systèmes, reconnaissance d'images*. Masson. http://books.google.it/books?id=uurXMgEACAAJ

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate (Y. Bengio & Y. LeCun, Eds.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. http://arxiv.org/abs/1409.0473

Ballesteros, M., Bohnet, B., Mille, S., & Wanner, L. (2015). Data-driven sentence generation with non-isomorphic trees (R. Mihalcea, J. Y. Chai, & A. Sarkar, Eds.). *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, 387–397. https://doi.org/10.3115/v1/n15-1042

Banaee, H., Ahmed, M. U., & Loutfi, A. (2013). Towards NLG for physiological data monitoring with body area networks (A. Gatt & H. Saggion,

Eds.). *ENLG 2013 - Proceedings of the 14th European Workshop on Natural Language Generation, August 8-9, 2013, Sofia, Bulgaria*, 193–197. https://aclanthology.org/W13-2127/

Banerjee, S., & Lavie, A. (2005). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 65–72. https://www.aclweb.org/anthology/W05-0909

Bengio, Y., Frasconi, P., & Simard, P. Y. (1993). The problem of learning long-term dependencies in recurrent networks. *Proceedings of International Conference on Neural Networks (ICNN'88), San Francisco, CA, USA, March 28 - April 1, 1993*, 1183–1188. https://doi.org/10.1109/ICNN.1993.298725

Bengio, Y., Simard, P. Y., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, *5*(2), 157–166. https://doi.org/10.1109/72.279181

Blum, A., Haghtalab, N., & Procaccia, A. D. (2015). Variational dropout and the local reparameterization trick [https://proceedings.neurips.cc/paper/2015/hash/bc7316929fe1545bf0b98d114ee3ecb8-Abstract.html]. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems 28: Annual conference on neural information processing systems 2015, december 7-12, 2015, montreal, quebec, canada* (pp. 2575–2583).

Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Leveling, J., Monz, C., Pecina, P., Post, M., Saint-Amand, H., Soricut, R., Specia, L., & Tamchyna, A. (2014). Findings of the 2014 workshop on statistical machine translation. *Proceedings of the Ninth Workshop on Statistical Machine Translation*, 12–58. https://doi.org/10.3115/v1/W14-3302

Bonetta, G., Cancelliere, R., Liu, D., & Vozila, P. (2021a). Retrieval-augmented transformer-xl for close-domain dialog generation. *The International FLAIRS Conference Proceedings*, *34*. https://doi.org/10.32473/flairs.v34i1.128369

Bonetta, G., Ribero, M., & Cancelliere, R. (2022). Regularization-based pruning of irrelevant weights in deep neural architectures. *Submitted*.

Bonetta, G., Roberti, M., Cancelliere, R., & Gallinari, P. (2021b). The rare word issue in natural language generation: A character-based solution. *Informatics*, *8*(1), 20. https://doi.org/10.3390/informatics8010020

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press. https://web.stanford.edu/~boyd/cvxbook/

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 1877–1901). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

Burke, R. D., Hammond, K. J., & Young, B. C. (1997). The findme approach to assisted browsing. *IEEE Expert*, *12*(4), 32–40. https://doi.org/10.1109/64.608186

Byrne, B., Krishnamoorthi, K., Sankar, C., Neelakantan, A., Goodrich, B., Duckworth, D., Yavuz, S., Dubey, A., Kim, K., & Cedilnik, A. (2019). Taskmaster-1: Toward a realistic and diverse dialog dataset. In K. Inui, J. Jiang, V. Ng, & X. Wan (Eds.), *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing, EMNLP-IJCNLP 2019, hong kong, china, november 3-7, 2019* (pp. 4515–4524). Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1459

Chen, B., & Cherry, C. (2014). A systematic comparison of smoothing techniques for sentence-level BLEU. *Proceedings of the Ninth Workshop on Statistical Machine Translation, WMT@ACL 2014, June 26-27, 2014, Baltimore, Maryland, USA*, 362–367. https://doi.org/10.3115/v1/w14-3346

Chen, D. L., & Mooney, R. J. (2008). Learning to sportscast: A test of grounded language acquisition (W. W. Cohen, A. McCallum, & S. T. Roweis, Eds.). *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, *307*, 128–135. https://doi.org/10.1145/1390156.1390173

Cheng, J., Dong, L., & Lapata, M. (2016). Long short-term memory-networks for machine reading. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 551–561. https://doi.org/10.18653/v1/D16-1053

Chisholm, A., Radford, W., & Hachey, B. (2017). Learning to generate one-sentence biographies from wikidata (M. Lapata, P. Blunsom, & A. Koller, Eds.). *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, 633–642. https://doi.org/10.18653/v1/e17-1060

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation (A. Moschitti, B. Pang, & W. Daelemans, Eds.). *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 1724–1734. http://aclweb.org/anthology/D/D14/D14-1179.pdf

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., & Weller, A. (2021). Rethinking attention with performers. *International Conference on Learning Representations*. https://openreview.net/forum?id=Ua6zuk0WRH

Collins, M. D., & Kohli, P. (2014). Memory bounded deep convolutional networks.

CommonCrawl. (2012). CommonCrawl's dataset. CommonCrawl. Retrieved January 1, 2012, from https://commoncrawl.org/

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., & Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, *abs/1901.02860*. http://arxiv.org/abs/1901.02860

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE conference on computer vision and pattern recognition*, 248–255.

Dept., C.-M. U. S. (2015). Speech understanding systems: summary of results of the five-year research effort at Carnegie-Mellon University. https://doi.org/10.1184/R1/6609821.v1

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR, abs/1810.04805*. http://arxiv.org/abs/1810.04805

Dhingra, B., Faruqui, M., Parikh, A., Chang, M.-W., Das, D., & Cohen, W. (2019). Handling divergent reference texts when evaluating table-to-text generation. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 4884–4895. https://doi.org/10.18653/v1/P19-1483

Dietrich, A., Gressmann, F., Orr, D., Chelombiev, I., Justus, D., & Luschi, C. (2021). Towards structured dynamic sparse pre-training of BERT. *CoRR, abs/2108.06277*. https://arxiv.org/abs/2108.06277

Doddington, G. (2002). Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. *Proceedings of the Second International Conference on Human Language Technology Research*, 138–145. http://dl.acm.org/citation.cfm?id=1289189.1289273

Duboué, P. A., & McKeown, K. R. (2003). Statistical acquisition of content selection rules for natural language generation. *Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP 2003, Sapporo, Japan, July 11-12, 2003*. https://aclanthology.org/W03-1016/

Dusek, O., & Jurcícek, F. (2016). Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*. https://doi.org/10.18653/v1/p16-2008

Dusek, O., Novikova, J., & Rieser, V. (2020). Evaluating the state-of-the-art of end-to-end natural language generation: The E2E NLG challenge. *Comput. Speech Lang.*, *59*, 123–156. https://doi.org/10.1016/j.csl.2019.06.009

Elhadad, M., McKeown, K. R., & Robin, J. (1997). Floating constraints in lexical choice. *Comput. Linguistics*, *23*(2), 195–239.

Engonopoulos, N., & Koller, A. (2014). Generating effective referring expressions using charts (M. Mitchell, K. F. McCoy, D. McDonald, & A. Cahill, Eds.). *INLG 2014 - Proceedings of the Eighth International Natural Language Generation Conference, Including Proceedings of the INLG and SIGDIAL 2014 Joint Session, 19-21 June 2014, Philadelphia, PA, USA*, 6–15. https://doi.org/10.3115/v1/w14-5002

Fan, A., Lewis, M., & Dauphin, Y. (2018). Hierarchical neural story generation. *ACL*.

Feng, Y., Ma, L., Liu, W., & Luo, J. (2019). Unsupervised image captioning. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Five9. (2021). Five9 blended contact center [The Five9 Blended Contact Center]. https://www.five9.com/products/blended-inbound-outbound-call-center

Gage, P. (1994). A new algorithm for data compression. *C Users J.*, *12*(2), 23–38.

Gale, T., Elsen, E., & Hooker, S. (2019). The state of sparsity in deep neural networks. *CoRR, abs/1902.09574*. http://arxiv.org/abs/1902.09574

Gardent, C., Shimorina, A., Narayan, S., & Perez-Beltrachini, L. (2017a). Creating training corpora for NLG micro-planners (R. Barzilay & M. Kan, Eds.). *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, 179–188. https://doi.org/10.18653/v1/P17-1017

Gardent, C., Shimorina, A., Narayan, S., & Perez-Beltrachini, L. (2017b). The webnlg challenge: Generating text from RDF data (J. M. Alonso, A. Bugarín, & E. Reiter, Eds.). *Proceedings of the 10th International Conference on Natural Language Generation, INLG 2017, Santiago de Compostela, Spain, September 4-7, 2017*, 124–133. https://doi.org/10.18653/v1/w17-3518

Gatt, A., & Krahmer, E. (2018). Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *J. Artif. Intell. Res.*, *61*, 65–170. https://doi.org/10.1613/jair.5477

Gatt, A., Portet, F., Reiter, E., Hunter, J., Mahamood, S., Moncur, W., & Sripada, S. (2009). From data to text in the neonatal intensive care unit: Using NLG technology for decision support and information management. *AI Commun.*, *22*(3), 153–186. https://doi.org/10.3233/AIC-2009-0453

Ge, T., He, K., Ke, Q., & Sun, J. (2013). Optimized product quantization for approximate nearest neighbor search. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. (2017). Convolutional sequence to sequence learning (D. Precup & Y. W. Teh, Eds.). *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, *70*, 1243–1252. http://proceedings.mlr.press/v70/gehring17a.html

Gers, F. A., Schmidhuber, J., & Cummins, F. A. (2000). Learning to forget: Continual prediction with LSTM. *Neural Comput.*, *12*(10), 2451–2471. https://doi.org/10.1162/089976600300015015

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks (G. J. Gordon, D. B. Dunson, & M. Dudík, Eds.). *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, *15*, 315–323. http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

Goldberg, E., Driedger, N., & Kittredge, R. I. (1994). Using natural-language processing to produce weather forecasts. *IEEE Expert*, *9*(2), 45–53. https://doi.org/10.1109/64.294135

Gomez, A. N., Zhang, I., Swersky, K., Gal, Y., & Hinton, G. E. (2019). Learning sparse networks using targeted dropout. *CoRR*, *abs/1905.13678*. http://arxiv.org/abs/1905.13678

Goodfellow, I. J., Bengio, Y., & Courville, A. (2016). *Deep learning* [http://www.deeplearningbook.org]. MIT Press.

Goyal, R., Dymetman, M., & Gaussier, É. (2016). Natural language generation through character-based rnns with finite-state prior knowledge (N. Calzolari, Y. Matsumoto, & R. Prasad, Eds.). *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan*, 1083–1092. http://aclweb.org/anthology/C/C16/C16-1103.pdf

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines [cite arxiv:1410.5401]. http://arxiv.org/abs/1410.5401

Gray, R., & Neuhoff, D. (1998). Quantization. *IEEE Transactions on Information Theory*, *44*(6), 2325–2383. https://doi.org/10.1109/18.720541

Gu, J., Lu, Z., Li, H., & Li, V. O. K. (2016). Incorporating copying mechanism in sequence-to-sequence learning (K. Erj & N. A. Smith, Eds.). *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. http://aclweb.org/anthology/P/P16/P16-1154.pdf

Guo, D., Tang, D., Duan, N., Zhou, M., & Yin, J. (2018). Dialog-to-action: Conversational question answering over a large-scale knowledge base. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/d63fbf8c3173730f82b150c5ef38b8ff-Paper.pdf

Guo, L., Liu, J., Zhu, X., Yao, P., Lu, S., & Lu, H. (2020). Normalized and geometry-aware self-attention network for image captioning. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Guo, Y., Yao, A., & Chen, Y. (2016). Dynamic network surgery for efficient dnns. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems 29: Annual conference on neural information processing systems 2016, december 5-10, 2016, barcelona, spain* (pp. 1379–1387). https://proceedings.neurips.cc/paper/2016/hash/2823f4797102ce1a1aec05359cc16dd9-Abstract.html

Guu, K., Hashimoto, T. B., Oren, Y., & Liang, P. (2017). Generating sentences by editing prototypes. *CoRR*, *abs/1709.08878*. http://arxiv.org/abs/1709.08878

Guu, K., Lee, K., Tung, Z., Pasupat, P., & Chang, M. (2020). REALM: retrieval-augmented language model pre-training. *CoRR*, *abs/2002.08909*. https://arxiv.org/abs/2002.08909

Han, S., Pool, J., Tran, J., & Dally, W. J. (2015). Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems 28: Annual conference on neural information processing systems 2015, december 7-12, 2015, montreal, quebec, canada* (pp. 1135–1143). https://proceedings.neurips.cc/paper/2015/hash/%5C%5Cae0eb3eed39d2bcef4622b2499a05fe6-Abstract.html

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition, 770–778. https://doi.org/10.1109/CVPR.2016.90

He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., & Li, M. (2019). Bag of tricks for image classification with convolutional neural networks. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Holtzman, A., Buys, J., Du, L., Forbes, M., & Choi, Y. (2020). The curious case of neural text degeneration. *International Conference on Learning Representations*. https://openreview.net/forum?id=rygGQyrFvH

Jarrett, K., Kavukcuoglu, K., Ranzato, M., & LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, 2146–2153. https://doi.org/10.1109/ICCV.2009.5459469

Johnson, J., Douze, M., & Jégou, H. (2017). Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*.

Kannan, A., Kurach, K., Ravi, S., Kaufmann, T., Tomkins, A., Miklos, B., Corrado, G., Lukacs, L., Ganea, M., Young, P., et al. (2016). Smart reply: Automated response suggestion for email. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 955–964.

Karpathy, A., & Li, F. (2015). Deep visual-semantic alignments for generating image descriptions. *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 3128–3137. https://doi.org/10.1109/CVPR.2015.7298932

Karpukhin, V., Oguz, B., Min, S., Wu, L., Edunov, S., Chen, D., & Yih, W. (2020). Dense passage retrieval for open-domain question answering. *CoRR*, *abs/2004.04906*. https://arxiv.org/abs/2004.04906

Kedzie, C., & McKeown, K. R. (2020). Controllable meaning representation to text generation: Linearization and data augmentation strategies (B. Webber, T. Cohn, Y. He, & Y. Liu, Eds.). *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, 5160–5185. https://doi.org/10.18653/v1/2020.emnlp-main.419

Khandelwal, U., Levy, O., Jurafsky, D., Zettlemoyer, L., & Lewis, M. (2020). Generalization through memorization: Nearest neighbor language models. *Proceedings of the 2020 International Conference on Learning Representations*. https://openreview.net/forum?id=HklBjCEKvH

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization (Y. Bengio & Y. LeCun, Eds.). *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. http://arxiv.org/abs/1412.6980

Koehn, P. (2005). Europarl: A Parallel Corpus for Statistical Machine Translation. *Conference Proceedings: the tenth Machine Translation Summit*, 79–86. http://mt-archive.info/MTS-2005-Koehn.pdf

Kore.ai. (2021). Ai-native contact center as a service [AI-native end-to-end Contact Center as-a-Service [CCaaS] solution]. https://kore.ai/smartassist/

Kudo, T., & Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, 66–71. https://doi.org/10.18653/v1/D18-2012

Lebret, R., Grangier, D., & Auli, M. (2016). Neural text generation from structured data with application to the biography domain (J. Su, X. Carreras, & K. Duh, Eds.). *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, 1203–1213. https://doi.org/10.18653/v1/d16-1128

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551. https://doi.org/10.1162/neco.1989.1.4.541

LeCun, Y., & Cortes, C. (1990). MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/

Lee, K., Chang, M., & Toutanova, K. (2019). Latent retrieval for weakly supervised open domain question answering. *CoRR*, *abs/1906.00300*. http://arxiv.org/abs/1906.00300

Lee, N., Ajanthan, T., & Torr, P. H. S. (2019). Snip: Single-shot network pruning based on connection sensitivity. *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. https://openreview.net/forum?id=B1VZqjAcYX

Leppänen, L., Munezero, M., Granroth-Wilding, M., & Toivonen, H. (2017). Data-driven news generation for automated journalism (J. M. Alonso, A. Bugarín, & E. Reiter, Eds.). *Proceedings of the 10th International Conference on Natural Language Generation, INLG 2017, Santiago de Compostela, Spain, September 4-7, 2017*, 188–197. https://doi.org/10.18653/v1/w17-3528

Li, J., Sun, A., Han, J., & Li, C. (2022). A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering*, *34*, 50–70.

Li, J., Galley, M., Brockett, C., Gao, J., & Dolan, B. (2016). A diversity-promoting objective function for neural conversation models. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 110–119. https://doi.org/10.18653/v1/N16-1014

Li, Y., Kaiser, L., Bengio, S., & Si, S. (2019). Area attention. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (pp. 3846–3855). PMLR.

Liang, P., Jordan, M. I., & Klein, D. (2009). Learning semantic correspondences with less supervision (K. Su, J. Su, & J. Wiebe, Eds.). *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*, 91–99. https://aclanthology.org/P09-1011/

Lin, C.-Y. (2004). ROUGE: A package for automatic evaluation of summaries. *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, 74–81. https://www.aclweb.org/anthology/W04-1013

Lin, Z., Huang, X., Ji, F., Chen, H., & Zhang, Y. (2019). Task-oriented conversation generation using heterogeneous memory networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 4558–4567. https://doi.org/10.18653/v1/D19-1463

Liu, D., Bonetta, G., Zhiping, F., Aron, D., Peter, S., & Joseph, V. P. (2021). System and method for generating responses for conversational agents [US patent n. 20-0007].

Liu, T., Luo, F., Xia, Q., Ma, S., Chang, B., & Sui, Z. (2019). Hierarchical encoder with auxiliary supervision for neural table-to-text generation: Learning better representation for tables. *AAAI*. https://doi.org/10.1609/aaai.v33i01.33016786

Liu, T., Wang, K., Sha, L., Chang, B., & Sui, Z. (2018). Table-to-text generation by structure-aware seq2seq learning (S. A. McIlraith & K. Q. Weinberger, Eds.). *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 4881–4888. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16599

Louizos, C., Welling, M., & Kingma, D. P. (2018). Learning sparse neural networks through l_0 regularization. *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. https://openreview.net/forum?id=H1Y8hhg0b

Luong, M., Le, Q. V., Sutskever, I., Vinyals, O., & Kaiser, L. (2016). Multi-task sequence to sequence learning. In Y. Bengio & Y. LeCun (Eds.), *4th international conference on learning representations, ICLR 2016, san juan, puerto rico, may 2-4, 2016, conference track proceedings*. http://arxiv.org/abs/1511.06114

Luong, M., & Manning, C. D. (2016). Achieving open vocabulary neural machine translation with hybrid word-character models (K. Erj & N. A. Smith, Eds.). *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. http://aclweb.org/anthology/P/P16/P16-1100.pdf

Luong, T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation (L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, & Y. Marton, Eds.). *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, 1412–1421. https://doi.org/10.18653/v1/d15-1166

Ma, D., Li, S., Zhang, X., & Wang, H. (2017). Interactive attention networks for aspect-level sentiment classification. *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 4068–4074.

Madotto, A., Wu, C., & Fung, P. (2018). Mem2seq: Effectively incorporating knowledge bases into end-to-end task-oriented dialog systems. *CoRR, abs/1804.08217*. http://arxiv.org/abs/1804.08217

Malkov, Y. A., & Yashunin, D. A. (2020). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*, 824–836.

Mei, H., Bansal, M., & Walter, M. R. (2016). What to talk about and how? selective generation using lstms with coarse-to-fine alignment (K. Knight, A. Nenkova, & O. Rambow, Eds.). *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, 720–730. http://aclweb.org/anthology/N/N16/N16-1086.pdf

Meteer, M. (1991). Bridging the generation gap between text planning and linguistic realization. *Computational Intelligence*, *7*.

Michel, P., Levy, O., & Neubig, G. (2019). Are sixteen heads really better than one? In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/2c601ad9d2ff9bc8b282670cdd54f69f-Paper.pdf

Microsoft. (2021). Microsoft cortana [The Microsoft Corporation virtual assistant (v.365.0.0.0, 28 febbraio 2020)]. www.apple.com/it/ios/siri/

Microsoft-Nvidia. (2022). Megatron-turing nlg 530b [The Megatron-Turing NLG 530B developed by Nvidia and Microsoft]. https://www.microsoft.com/en-us/research/blog/using-deepspeed-and-megatron-to-train-

megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., & Khudanpur, S. (2010). Recurrent neural network based language model (T. Kobayashi, K. Hirose, & S. Nakamura, Eds.). *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, 1045–1048. http://www.isca-speech.org/archive/interspeech%5C_2010/i10%5C_1045.html

Miller, G. A. (1995). Wordnet: A lexical database for english. *Commun. ACM*, *38*(11), 39–41. https://doi.org/10.1145/219717.219748

Molchanov, D., Ashukha, A., & Vetrov, D. P. (2017). Variational dropout sparsifies deep neural networks. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning, ICML 2017, sydney, nsw, australia, 6-11 august 2017* (pp. 2498–2507). PMLR. http://proceedings.mlr.press/v70/molchanov17a.html

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines (J. Fürnkranz & T. Joachims, Eds.). *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, 807–814. https://icml.cc/Conferences/2010/papers/432.pdf

Novikova, J., Dusek, O., & Rieser, V. (2017). The E2E dataset: New challenges for end-to-end generation (K. Jokinen, M. Stede, D. DeVault, & A. Louis, Eds.). *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, Saarbrücken, Germany, August 15-17, 2017*, 201–206. https://doi.org/10.18653/v1/w17-5525

O'Donnell, M., Mellish, C., Oberlander, J., & Knott, A. (2001). ILEX: an architecture for a dynamic hypertext generation system. *Nat. Lang. Eng.*, *7*(3), 225–250. https://doi.org/10.1017/S1351324901002698

Otter, D. W., Medina, J. R., & Kalita, J. K. (2018). A survey of the usages of deep learning in natural language processing. *CoRR*, *abs/1807.10854*. http://arxiv.org/abs/1807.10854

Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002). Bleu: A method for automatic evaluation of machine translation. https://doi.org/10.3115/1073083.1073135

Parikh, A., Täckström, O., Das, D., & Uszkoreit, J. (2016). A decomposable attention model for natural language inference. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2249–2255. https://doi.org/10.18653/v1/D16-1244

Parikh, A. P., Wang, X., Gehrmann, S., Faruqui, M., Dhingra, B., Yang, D., & Das, D. (2020). Totto: A controlled table-to-text generation dataset (B. Webber, T. Cohn, Y. He, & Y. Liu, Eds.). *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, 1173–1186. https://doi.org/10.18653/v1/2020.emnlp-main.89

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, *28*, 1310–1318. http://jmlr.org/proceedings/papers/v28/pascanu13.html

Perez-Beltrachini, L., & Gardent, C. (2017). Analysing data-to-text generation benchmarks. *INLG*. https://www.aclweb.org/anthology/W17-3537.pdf

Plachouras, V., Smiley, C., Bretz, H., Taylor, O., Leidner, J. L., Song, D., & Schilder, F. (2016). Interacting with financial data using natural language (R. Perego, F. Sebastiani, J. A. Aslam, I. Ruthven, & J. Zobel, Eds.). *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, 1121–1124. https://doi.org/10.1145/2911451.2911457

Puduppully, R., Dong, L., & Lapata, M. (2019a). Data-to-text generation with content selection and planning. *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, 6908–6915. https://doi.org/10.1609/aaai.v33i01.33016908

Puduppully, R., Dong, L., & Lapata, M. (2019b). Data-to-text generation with entity modeling (A. Korhonen, D. R. Traum, & L. Màrquez, Eds.). *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, 2023–2035. https://doi.org/10.18653/v1/p19-1195

Ramos-Soto, A., Diz, A. J. B., Barro, S., & Taboada, J. (2015). Linguistic descriptions for automatic generation of textual short-term weather forecasts on real prediction data. *IEEE Trans. Fuzzy Syst.*, *23*(1), 44–57. https://doi.org/10.1109/TFUZZ.2014.2328011

Ranzato, M., Chopra, S., Auli, M., & Zaremba, W. (2016). Sequence level training with recurrent neural networks. In Y. Bengio & Y. LeCun (Eds.), *4th international conference on learning representations, ICLR 2016, san juan, puerto rico, may 2-4, 2016, conference track proceedings*. http://arxiv.org/abs/1511.06732

Rebuffel, C., Roberti, M., Soulier, L., Scoutheeten, G., Cancelliere, R., & Gallinari, P. (2021). Controlling hallucinations at word level in data-to-text generation. *CoRR*, *abs/2102.02810*. https://arxiv.org/abs/2102.02810

Reiter, E. (1994). Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? *Proceedings of the Seventh International Workshop on Natural Language Generation, INLG 1994, Kennebunkport, Maine, USA, June 21-24, 1994*. https://aclanthology.org/W94-0319/

Reiter, E. (2010). Natural language generation. *The handbook of computational linguistics and natural language processing* (pp. 574–598). John Wiley & Sons, Ltd. https://doi.org/https://doi.org/10.1002/9781444324044.ch20

Reiter, E., & Dale, R. (1997). Building applied natural language generation systems. *Nat. Lang. Eng.*, *3*(1), 57–87. https://doi.org/10.1017/S1351324997001502

Reiter, E., Sripada, S., Hunter, J., Yu, J., & Davy, I. (2005). Choosing words in computer-generated weather forecasts. *Artif. Intell.*, *167*(1-2), 137–169. https://doi.org/10.1016/j.artint.2005.06.006

Roberti, M., Bonetta, G., Cancelliere, R., & Gallinari, P. (2019). Copy mechanism and tailored training for character-based data-to-text generation (U. Brefeld, É. Fromont, A. Hotho, A. J. Knobbe, M. H. Maathuis, & C. Robardet, Eds.). *Machine Learning and Knowledge Discovery in Databases*

- European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part II, *11907*, 648–664. https://doi.org/10.1007/978-3-030-46147-8_39

Ruder, S., Peters, M. E., Swayamdipta, S., & Wolf, T. (2019). Transfer learning in natural language processing. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, 15–18. https://doi.org/10.18653/v1/N19-5004

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. *Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1: Foundations* (pp. 318–362). MIT Press.

Salehinejad, H., & Valaee, S. (2020). Edropout: Energy-based dropout and pruning of deep neural networks. *CoRR*, *abs/2006.04270*. https://arxiv.org/abs/2006.04270

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.*, *45*(11), 2673–2681. https://doi.org/10.1109/78.650093

See, A., Liu, P. J., & Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. *ACL*. https://doi.org/10.18653/v1/P17-1099

Sennrich, R., Haddow, B., & Birch, A. (2016). Neural machine translation of rare words with subword units (K. Erj & N. A. Smith, Eds.). *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. http://aclweb.org/anthology/P/P16/P16-1162.pdf

Serban, I., Sordoni, A., Bengio, Y., Courville, A., & Pineau, J. (2016). Building end-to-end dialogue systems using generative hierarchical neural network models. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016)*, 3776–3784. https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11957

Serban, I. V., Klinger, T., Tesauro, G., Talamadupula, K., Zhou, B., Bengio, Y., & Courville, A. (2017). Multiresolution recurrent neural networks: An application to dialogue response generation. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*.

Sha, L., Mou, L., Liu, T., Poupart, P., Li, S., Chang, B., & Sui, Z. (2018). Order-planning neural text generation from structured data (S. A. McIlraith & K. Q. Weinberger, Eds.). *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 5414–5421. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16203

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*.

Sordoni, A., Bachman, P., & Bengio, Y. (2016). Iterative alternating neural attention for machine reading. *CoRR*, *abs/1606.02245*. http://arxiv.org/abs/1606.02245

Stock, O., Zancanaro, M., Busetta, P., Callaway, C. B., Krüger, A., Kruppa, M., Kuflik, T., Not, E., & Rocchi, C. (2007). Adaptive, intelligent presentation of information for the museum visitor in PEACH. *User Model. User*

Adapt. Interact., *17*(3), 257–304. https://doi.org/10.1007/s11257-007-9029-6

Sukhbaatar, S., szlam arthur, a., Weston, J., & Fergus, R. (2015). End-to-end memory networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems 28* (pp. 2440–2448). Curran Associates, Inc. http://papers.nips.cc/paper/5846-end-to-end-memory-networks.pdf

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger, Eds.). *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 3104–3112. https://proceedings.neurips.cc/paper/2014

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Tartaglione, E., Bragagnolo, A., Fiandrotti, A., & Grangetto, M. (2022). Loss-based sensitivity regularization: Towards deep sparse neural networks. *Neural Networks*, *146*, 230–237. https://doi.org/10.1016/j.neunet.2021.11.029

Tartaglione, E., Lepsøy, S., Fiandrotti, A., & Francini, G. (2018). Learning sparse neural networks via sensitivity-driven regularization (S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett, Eds.). *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 3882–3892. https://proceedings.neurips.cc/paper/2018/hash/04df4d434d481c5bb723be1b6df1ee65-Abstract.html

Theune, M., Klabbers, E., de Pijper, J., Krahmer, E., & Odijk, J. (2001). From data to speech: A general approach. *Nat. Lang. Eng.*, *7*(1), 47–86. http://journals.cambridge.org/action/displayAbstract?aid=73673

Tiedemann, J. (2012). Parallel data, tools and interfaces in opus. In N. C. (Chair), K. Choukri, T. Declerck, M. U. Dogan, B. Maegaard, J. Mariani, J. Odijk, & S. Piperidis (Eds.), *Proceedings of the eight international conference on language resources and evaluation (lrec'12)*. European Language Resources Association (ELRA).

Tikhonov, A. N. (1963). Solution of incorrectly formulated problems and the regularization method. *Soviet Math. Dokl.*, *4*, 1035–1038.

Tikhonov, A., & Arsenin, V. (1977). *Solutions of ill-posed problems*. Winston, Washington DC. http://books.google.co.in/books?id=ECrvAAAAMAAJ

Torralba, A., Fergus, R., & Freeman, W. T. (2008). 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *30*(11), 1958–1970. https://doi.org/10.1109/TPAMI.2008.128

Tudor Car, L., Dhinagaran, D. A., Kyaw, B. M., Kowatsch, T., Joty, S., Theng, Y.-L., & Atun, R. (2020). Conversational agents in health care: Scoping review and conceptual analysis. *J Med Internet Res*, *22*(8). https://doi.org/10.2196/17158

Turner, R., Sripada, S., Reiter, E., & Davy, I. P. (2007). Selecting the content of textual descriptions of geographically located events in spatio-temporal

weather data (R. Ellis, T. Allen, & M. Petridis, Eds.). *Applications and Innovations in Intelligent Systems XV - Proceedings of AI-2007, the Twenty-seventh SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Cambridge, UK, 10-12 December 2007*, 75–88. https://doi.org/10.1007/978-1-84800-086-5_6

Ullrich, K., Meeds, E., & Welling, M. (2017). Soft weight-sharing for neural network compression. *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. https://openreview.net/forum?id=HJGwcKclx

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. https://arxiv.org/pdf/1706.03762.pdf

Vedantam, R., Zitnick, C. L., & Parikh, D. (2015). Cider: Consensus-based image description evaluation. *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, 4566–4575. https://doi.org/10.1109/CVPR.2015.7299087

Vinyals, O., & Le, Q. (2015). A neural conversational model. *Proceedings of ICML Deep Learning Workshop*.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., & Titov, I. (2019). Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 5797–5808. https://doi.org/10.18653/v1/P19-1580

Wen, T., Gasic, M., Kim, D., Mrksic, N., Su, P., Vandyke, D., & Young, S. J. (2015a). Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. *Proceedings of the SIGDIAL 2015 Conference, The 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue, 2-4 September 2015, Prague, Czech Republic*, 275–284. https://doi.org/10.18653/v1/w15-4639

Wen, T., Gasic, M., Mrksic, N., Su, P., Vandyke, D., & Young, S. J. (2015b). Semantically conditioned lstm-based natural language generation for spoken dialogue systems (L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, & Y. Marton, Eds.). *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, 1711–1721. https://doi.org/10.18653/v1/d15-1199

Werbos, P. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560. https://doi.org/10.1109/5.58337

Weston, J., Chopra, S., & Bordes, A. (2014). Memory networks [cite arxiv:1410.3916]. http://arxiv.org/abs/1410.3916

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, *1*(2), 270–280. https://doi.org/10.1162/neco.1989.1.2.270

Wiseman, S., Shieber, S. M., & Rush, A. M. (2017). Challenges in data-to-document generation (M. Palmer, R. Hwa, & S. Riedel, Eds.). *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, 2253–2263. https://doi.org/10.18653/v1/d17-1239

Wiseman, S., Shieber, S. M., & Rush, A. M. (2018). Learning neural templates for text generation (E. Riloff, D. Chiang, J. Hockenmaier, & J. Tsujii, Eds.). *Proceedings of the 2018 Conference on Empirical Methods in Natural*

Language Processing, Brussels, Belgium, October 31 - November 4, 2018, 3174–3187. https://doi.org/10.18653/v1/d18-1356

Wolf, T., Sanh, V., Chaumond, J., & Delangue, C. (2019). Transfertransfo: A transfer learning approach for neural network based conversational agents. *CoRR*, *abs/1901.08149*. http://arxiv.org/abs/1901.08149

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., … Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, *abs/1609.08144*.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *CoRR*, *abs/1708.07747*. http://arxiv.org/abs/1708.07747

Xing, C., Wu, Y., Zhou, M., Huang, Y., & Ma, W.-Y. (2018). Hierarchical recurrent attention network for response generation. *Proceedings of the The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, 5610–5617.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J. G., Salakhutdinov, R., & Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, *abs/1906.08237*. http://arxiv.org/abs/1906.08237

Yang, Z., Blunsom, P., Dyer, C., & Ling, W. (2017). Reference-aware language models (M. Palmer, R. Hwa, & S. Riedel, Eds.). *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, 1850–1859. https://doi.org/10.18653/v1/d17-1197

Zhang, K., Gool, L. V., & Timofte, R. (2020). Deep unfolding network for image super-resolution. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhang, S., Dinan, E., Urbanek, J., Szlam, A., Kiela, D., & Weston, J. (2018). Personalizing dialogue agents: I have a dog, do you have pets too? *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2204–2213. https://doi.org/10.18653/v1/P18-1205

Zhang, Y., Sun, S., Galley, M., Chen, Y.-C., Brockett, C., Gao, X., Gao, J., Liu, J., & Dolan, B. (2020). DIALOGPT : Large-scale generative pre-training for conversational response generation. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 270–278. https://doi.org/10.18653/v1/2020.acl-demos.30

Zhang, Z., Han, X., Liu, Z., Jiang, X., Sun, M., & Liu, Q. (2019). ERNIE: enhanced language representation with informative entities. *CoRR*, *abs/1905.07129*. http://arxiv.org/abs/1905.07129