POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master degree course in Computer engineering

Master Degree Thesis

Automated matches summary generation

Automated text generation to create matches summaries given stats

Supervisors
prof. Paolo Garza

Candidates
Clemente Cetera
matricola: 257234

Internship Tutor
dott. Pietro Marini

Academic year 2020-2021

Summary

two models for automated text generation are created by the candidate in order to summarize top soccer European league matches giving as input statistics of the two teams collected during a game.
the first approach is obtained from a synthetic text generator, transformer-based language model: GPT-2 from OpenAI, used as an actual text generator: giving as input only few words about one match the model has to be able to create a summary in fluent, syntactically and grammatically correct English language. another way is then taken using the Text-to-text-transfer-transformer network (T5) by Google, pre-trained on the C4 dataset: the model, originally thought as a simple translator is used in this case as text generator to which the stats of the match are given as input in order to produce a summary of the game.
For legal and privacy reasons in all the examples and stats shown in this work names and personal information are replaced by generic nouns.

Acknowledgements

First and foremost thanks to my parents for their love, their caring and their sacrifices, none of this could be possible without a wonderful father and a terrific mother.
Thanks to my sister, the beautiful and safe harbour in which there is no need to be always moored since you know it will always be there when you need it.
I would like to express my deep and sincere gratitude to professor Garza for letting me have the opportunity to do this research but more important to have been a kind and inspiring example of what I would like to do and to be in my life.
Thanks to Deltatre and all the wonderful people met in the company, first for the opportunity of research but mainly for having been a technical and moral support in these months.
Thanks to all the professors and assistants met in these years, from the gentlest to the toughest, each of which, in their way, has taught me something I will never forget.
Thanks to my grandparents for teaching me that, no matter how hard life could hit, there is always a good reason to smile.
Thanks to my whole family, uncles and cousins, for the laughs and the breaks during difficult periods.
Thanks to all the friends that I met and the ones I have not met yet, life is a long journey, but amazing if you have wonderful people to travel with.

Contents

List of Tables

List of Figures

Part I

Problem Description and used architectures

Chapter 1

From LSTM to transformers

1.1 NLP: Natural language processing

In the wide world of artificial intelligence, Natural language processing is the field that has the difficult task to learn how to read, write, extract meaning and understand (both grammatically and semantically) the human language. An NLP model built from scratch can be thought as a child that owns in his mind a lot of words stored in his brain after having listened or read them (in our case, the vocabulary given to the network) but with not yet the experience and knowledge to understand how each word has to be put after or before another one in order to give to the sentence a specific meaning, how specific words preceded by others change their meaning or how to read a set of words catching its meaning and which feeling it is representing.
Natural language processing is one of the most exciting tasks nowadays (speech recognition and generation devices such as Alexa by Amazon and Siri by Apple are now widespread in the world) but also one of the most difficult for different reasons: first of all, the input data that have to be processed are words in the best case or, worse, entire sentences and/or articles. This means that the kind of information available is unstructured and so it can not be represented as a classical table or relational database. Another important feature that increases the difficulty to manage human language is that each word has not to be analyzed as a single token independent from the others but its meaning can completely change if the word is followed or preceded by other tokens. This means that during the analysis we can

not focus on a single word but also on the context in which that word is used.

1.2 Recurrent neural network and LSTM

As said before, one of the most tricky problem of NLP is that each word can depend more or less from the other words in the sample so a neural network has to take into account the words that have preceded the current token and this is not possible by using a convolutional neural network since it not only requires a fixed size input but also has no memory of its previous states, that's why the focus was switched in the '90s to recurrent neural networks (RNN).
RNNs change the paradigm of feed forward networks (the input of a specific layer can only comes from previous ones) by adding as input, besides previous outputs, also the hidden states of one or several neurons: in this way, in a specific time stamp t each neuron can have a composite input made by the previous outputs and some historical information about what happened at time $T < t$ (as shown in figure 1.1), so recurrent neural networks become useful for NLP since their intermediate values (states) can store information about past inputs (words) for a not fixed time.

However, Two big problems affect the power and accuracy of RNNs.
The first issue is represented by the exploding gradient problem that appears when the weights used in the network are big, leading the gradients to assume very large values causes of overflow errors, this kind of problem can still be solved partially by deciding a limit value for the gradient or artificially reducing it. The actual big problem in RNN is the vanishing gradient one that occurs when the weights used are small and during back-propagation weights belonging to the first layers are never updated.
The problem of vanishing gradient led to a new type of recurrent neural network: long short-term memory models (LSTM).
LSTM solves the vanishing gradient problem by adding a new structure in RNNs representing a memory cell: each memory cell has a single self-connected recurrent link of fixed weight one which ensure gradients to not change while flowing through several layers. Figure 1.2 represents a node in a LSTM and its unfolded version to represent how hidden states are propagated through time.

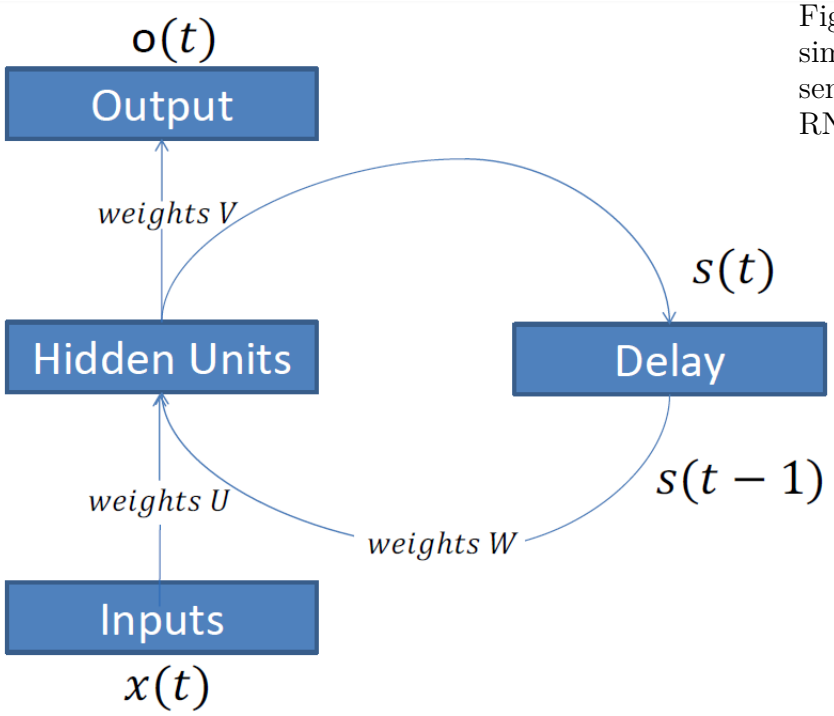


Figure 1.1. simplified representation of an RNN

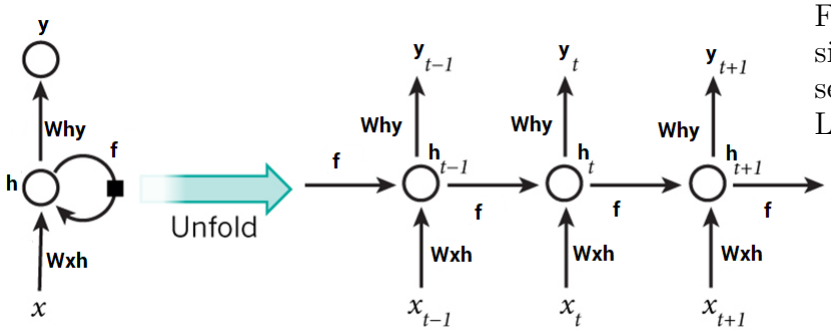


Figure 1.2. simplified representation of an LSTM

1.3 Transfomers

LSTM, because of its ability to remember (or forget) specific states based on the fact if one word is considered important (or not important), was for a long time the state of the art for what concerns NLP.
However in 2017, the paper "Attention is all you need" [1] published a new kind of model able to outperform long short-term memory architectures by describing two new concepts: Transformers and sequence to sequence

(seq2seq) architecture. As the name suggests seq2seq architectures are networks able to transform a specific sequence of data (e.g. a sequence of words) in another one. Like LSTM, seq2seq networks are very good in NLP tasks such as translation and text generation but the way they work is very different from previous models, in fact these kind of architectures are composed of two main characters: an encoder, that has the task of taking the input sequence and mapping it to an higher dimensional space representation, and a decoder whose task is to take the output representation of the encoder and transform it in an output sequence.

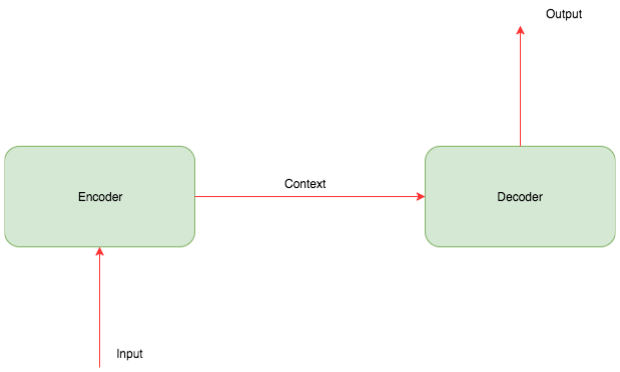


Figure 1.3. simplified representation of an Encoder-decoder based architecture

Figure 1.3 shows a simplified encoder-decoder based architecture: talking about the task of translation the encoder and decoder can be thought as two people that have to complete the task of translating a text from German to Italian but one of the two (encoder) can only speak German and English while the other one is able only to manage English and Italian languages. To perform the task, encoder translate the German text in English than the Decoder takes the English output of the first translator transforming it in the final Italian text.
Transformers follow the same architecture of seq2seq networks but introducing a brand new concept: Attention. Attention can be described as a function whose task is to map the input represented by three kind of vectors (queries, key-value pairs) to an output generated by the weighted sum of the values with the weight of each value given as output by a function that takes as input a query and the corresponding key. While reading or writing some text a human reads or writes a specific word thinking also at the context of that word in the sentence, the attention mechanism works in a similar way allowing the transformer not only to complete the task of language processing looking at the current word given as input but it also gives decoder the

possibility to know which of the previous words had more importance in the generation of the text.
Since both the models used for automated text generation in this case of study base their architecture on the original structure of transformer, a closer look to transformers architectures is needed. In details, transformers follow the base encoder-decoder architecture but using several number of modules: this innovative model uses stacked self-attention, fully-connected layers of encoder and decoders. With the term "self-attention" is indicated a layer very similar to the concept of "attention" described above but this time all the words in the input sequence are taken by the layer that produces the importance of each word in the text (in the example of translating a text from German to Italian, the self-attention mechanism can be thought as the first translator that underlines the most important words in the input text).

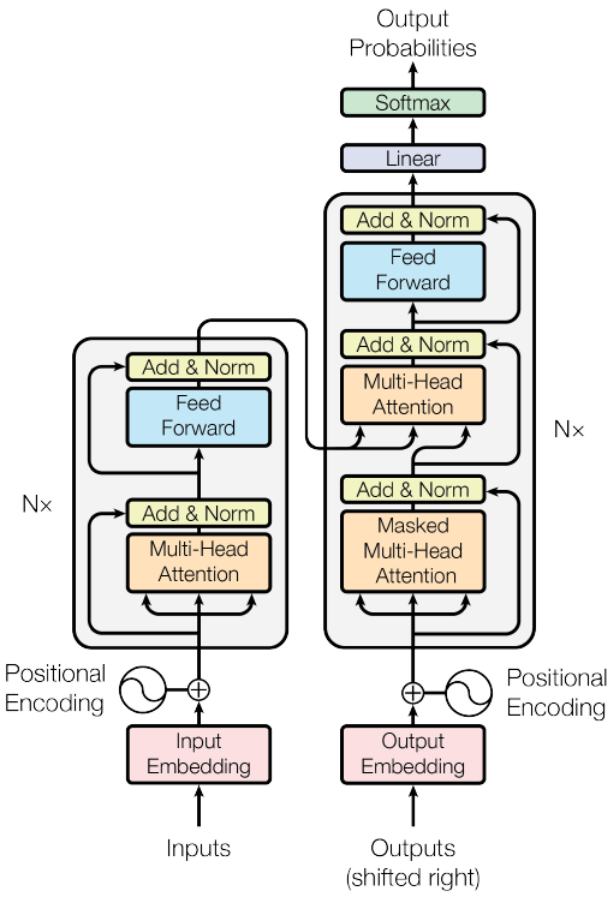


Figure 1.4. detailed representation of a transformer architecture

A detailed representation of transformers architecture is shown in figure 1.4: Encoder and decoder stacks module are represented respectively on the

left and right side of the picture. The label "Nx" is put near the two blocks to stress the concept of having not one single encoder or decoder module but several number of them. As it can be seen looking at the picture, modules are composed of Multi-headed attention and feed-forward layers. Given the input consisting of queries (aggregated in a matrix indicated with Q) and keys (matrix K) of dimension d_k and values (matrix V) with dimension d_v the attention function gives as output the result of the softmax function applied on the dot product queries and keys each divided by $\sqrt{d_k}$

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1.1}$$

The multi-headed attention is performed by projecting queries, values and keys h times and applying the function described above in parallel obtaining d_v dimensional values then aggregated and projected again to obtain the final values.
In the end, it's reasonable to assert that, through the multi-headed mechanism transformers apply the attention technique several times by linearly projecting the input vectors allowing the model to learn from different representation queries, values and keys.

$$MultiHead(Q; K; V) = Concat(head1; ...; headh)W^O \tag{1.2}$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$

The way input vectors are projected is shown in equation 1.2 where W represents the weight matrices learned during training.

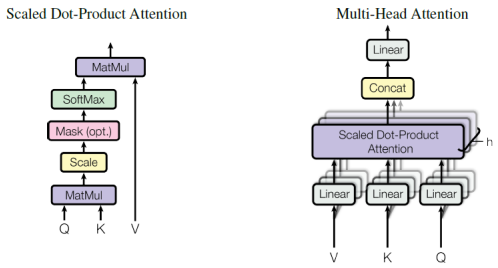


Figure 1.5. Scaled Dot-Product and multi-headed attention

In figure 1.5 a detailed representation of single-headed and multi-headed attention is shown, the second one allows the model to retrieve information

from different sub-spaces of input. The other important layer which builds the encoder and decoder is the position-wise feed-forward network consisting of two linear transformation with a RELU activation function between them.

Chapter 2

Transformer-based architectures

2.1 Generative Pre-Training Networks

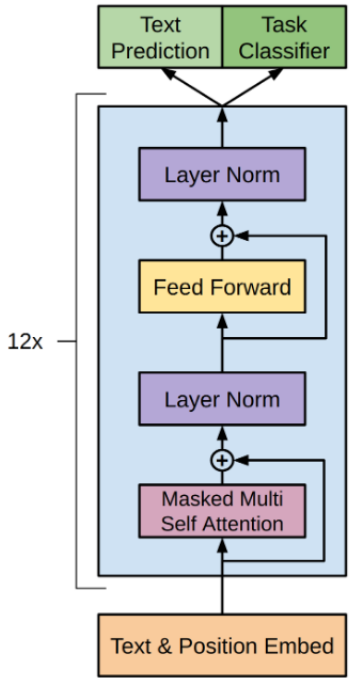
in 2018 A.Radford in his paper "Improving Language Understanding by Generative Pre-Training" [2] showed a new approach for NLP: in natural language processing the availability of unlabeled large text is quite huge while labeled data for specific NLP tasks are not so large. The idea born with GPT (Generative pre-training network) is to pre-train a model giving as input a very large amount of unlabeled text and then use the same network only fine-tuning it on a specific task with labeled data. This brand new semi-supervised approach shows very significant improvements in natural language processing tasks.
In details, The training phase of GPT consists in two different stages: the first one has the task to train a model with a huge amount of unlabeled text (of different nature, to stress the generalization and regularization of the model) followed then by the second phase in which a specific task (with labeled data) is performed.

A detailed description of the GPT architecture is shown in figure 2.1. In the unsupervised training phase, given a set of tokens $U = u_1, ...u_n$, the task consists in maximizing the likelihood:

$$L_1(U) = \sum_i \log P(u_i \mid u_{i-k}, ..., u_{i-1}; \Theta) \tag{2.1}$$

where k is the size of the context window (number of tokens considered

Figure 2.1. detailed representation of GPT architecture



in the current step) and P the conditional probability calculated using the parameters of the network θ.
In the case of Generative Pre-Training networks, the architecture is very similar to standard transformers but with a main difference: this kind of models are composed only of multi-layer decoders (so there is not the stack of encoders showed previously talking about original transformers architecture) and, as seen before, each decoder is composed by a multi-headed self-attention mechanism followed by a position-wise feed-forward layer in order to give as result an output probability distribution over each token. Decided the state of the first hidden layer, each following state can be described using the preceding one and all the hidden states are finally used in the computation of the final distribution:

$$h_0 = UW_e + W_p$$

$$h_l = transformer\_block(h_{l-1}) \forall i \in [1, n]$$

$$P(u) = softmax(h_n W_e^T) \tag{2.2}$$

where U = $(u_k,.....u_1)$ is the context vector of tokens, n is the number of layers,W_e is the token embedding matrix, and W_p is the position embedding matrix.
After the unsupervised task performed in order to maximize the likelihood function 2.1, a labeled dataset C has to be given as input for the discriminating phase: assuming that each sample of C is composed by a sequence of input tokens $x^1, ...x^m$ with the corresponding label y, these vectors are used in the unsupervised pre-trained network to obtain the final transformer's block activation h_l^m which is given as input with the parameters W_y to a linear output layer to predict y:

$$P(y \mid x^1, ..., X^m) = softmax(h_l^m W_y) \tag{2.3}$$

Finally, this gives a new objective function to be maximized:

$$L_2(C) = \sum_{x,y} \log P(y \mid x^1, ..., x^m) \tag{2.4}$$

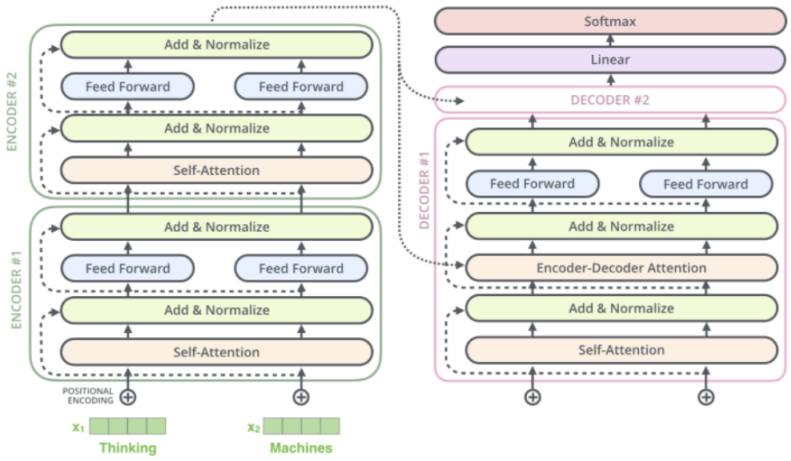
An improvement of the first generative pre-training model that is used in this case study, called GPT-2[3], has the same architecture described in picture 2.1 but with few modifications: the normalization layers are now moved to the input of all the sub-blocks and also an additional layer normalization is added after the final self-attention block, vocabulary, context and batch-size are increased and Also a initialization different from the one adopted in the previous model is performed. In the GPT-2 initialization the weights of residual layers are scaled by a factor of $\frac{1}{\sqrt{N}}$ with N representing the total number of residual layers.

2.2 Text-to-Text Transfer Transformer (T5)

In 2019, Colin Raffel et. al. in their paper "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer" [4] created a brand new transfomer-based model stressing to its limit the concept of transfer learning in order to build a multi-tasking architecture pre-trained with a huge NLP dataset (a detailed description of data used during T5 training is presented afterwards).
T5 model is based on the idea of treating every text processing problem as a "text-to-text" problem.

Looking at the picture representing the architecture of T5(figure 2.2), text-to-text transfer transformer model appears very similar to a standard

Figure 2.2. detailed representation of T5 architecture



transformer but with little differences. As a standard encoder-decoder transformer, T5 takes as input a sequence of tokens that are first embedded in a new representation and then passed to the encoder stack. As in original transformers, the encoder is composed by several blocks each of which includes a self-attention layer and a feed-forward network with a normalization layer[5] located right before the input of each block component but, in this case, the activations passing through normalization layers are only rescaled with no bias added.
The T5 decoder follows the same architecture of encoder but including this time a standard deviation mechanism after each self-attention layer in order to take into accounts encoder outputs.
The output of the last encoder block is given as input to a fully-connected layer with a softmax output that shares its weights with the input embedding matrix.
Another important feature that characterised text-to-text transfer transformer is that, while in standard transformers architecture are used fixed positional embeddings to provide an explicit positional signal of each token to the model, T5 is based on relative position embeddings to produce a different learned embedding related to the current offset between the key and the query that are being compared in the self-attention mechanism and this is done by adding a scalar to the corresponding logit used to calculate the attention weights.
Finally, T5 can be seen as a standard transformer architecture where the normalization layer is modified, and a different position embedding scheme is used.

2.3 Decoding strategies

Both GPT-2 and T5 are based on "auto-regressive" generation: after each token is produced, it is added to a sequence of tokens that becomes the next input of the model in its next step. More precisely, this text generation technique starts from the assumption that the probability distribution of the next word to be produce can be seen as the product of the conditional probabilities belonging to the tokens that precede the current one in the input sequence:

$$P(w_{1:T} \mid W_0) = \prod_{t=1}^{T} P(w_t \mid w_{1:t-1}, W_0) \text{ with } w_{1:0} = 0 \tag{2.5}$$

W_0 represents in this case the current sequence of tokens, T represents the length of the sequence to be generated and it is decided during decoding phase or as input parameter of the network (maximum number of words to be generated).
So it is simple to understand that the way transformers choose the words to be decoded has a big impact on the entire text generation mechanism.
A list of decoding strategies are now presented with respectively pros and cons in order to have a good understanding about which technique has to be chosen for each case of study.
The first decoding strategy presented is the "Greedy Search": this technique chooses the next word as the one with highest probability at each step.

$$w_t = \underset{w}{\operatorname{argmax}} P(w \mid w_{1:t-1}) \tag{2.6}$$

As classic greedy algorithms the biggest drawback in this kind of approach is that only the best probability at each step is chosen so no optimal paths are searched (e.g: the words that will lead to the best result overall preceded by a word with a very low probability will never be chosen).
to limit the problems deriving from the first technique, another approach, called "Beam search", can be used: it is very similar to the first one but in this case a number of steps equals to the number of beams given as input is analyzed. It appears clear that, even if beam search will always lead to a better knowledge of the space of tokens so to a better solution than the pure greedy approach, there is no guarantee that it will find the best output.
Rather than search the best solution by combining all the possible words in the vocabulary (exhaustive approach, not possible for computational and time reasons), a good way to increase the chance to choose one of the best

possible path to obtain results is to introduce randomness in the choice of the words to produce: the basic way to do this could be to randomly select a word at each step with its probability $P(w \mid w_{1:t-1})$ but this leads to very random results produced without taking into account what the path chosen is indicating with previous words.
The best choice to introduce randomness as decoding strategy is to pick the next word increasing the chances to choose a word with high probability by using the so called "Temperature" of the softmax layer: given the original formula of the softmax function

$$softmax(x)_i = \frac{e^{y_i}}{\sum_{j=1}^{N} e^{y_j}} \tag{2.7}$$

temperature represents an hyper-parameter to control the randomness of probabilities by scaling logits right before applying the softmax function

$$softmax(x)_i = \frac{e^{\frac{y_i}{T}}}{\sum_{j=1}^{N} e^{\frac{y_j}{T}}} \tag{2.8}$$

Setting T (temperature) equals to 1 represents the specific case in which no scaling is performed while, for example, with T=0.7 all the logits become larger increasing the value distance of resulting probabilities making more likely to pick words with higher probabilities in random sampling. setting temperature values very close to zero will lead to the pure greedy strategy seen before with the problems explained above.
Once it is established that pseudo-random picking next words represents the best choice, it is possible to combine the sampling and beam search approaches to produce for sure the best decoding strategies for text generation: in the first one, top-K sampling,the K most likely next words represent the set in which the random sampling is applied while in the second one, top-P sampling, instead of picking only between a set of k elements, the next word is chosen between the smallest possible set of tokens whose cumulative probability is greater than the parameter p.

Part II

Proposed solutions

Chapter 3

GPT-2 for text generation

3.1 Operations and original training

Looking at the architecture in section 2.1, it is not still clear how GPT-2 can create text from scratch so a simple representation of how this network chooses words to be added in a sequence is needed.
The main concept on which GPT-2 is based is that in the process of generating text, given an input token, only the next word is predicted in a step: for example, given as input the word "the", the following step in the network would be to choose the word in its own vocabulary that has the highest probability. In few words, at each step the network takes one input token to produce one output probable next token but this way to work appears not so aligned with what has been explained in section 1.3, with the self-attention mechanism that should work with vectors of queries and keys. GPT-2 exploits the multi-headed self-attention mechanism by taking into account all the previous predicted token: in a generic step, each decoder layer has always a single key-input vector pair but this pair will be concatenated with all the other pairs belonging to the same layer in previous step making self-attention work properly.
Finally, a summary of how GPT-2 creates text is the following: first, the input sequence of word is converted in a sequence of embeddings through Byte Pair Encoding [6] that is then given as input to the decoder that produces a sequence of output embeddings from which the first output word is sampled. The first output word is then embedded and given as input with index length-1 to the next step where a new single output word is created and given embedded as the length-2 input and so on. Actually the first token from which the process starts is a keyword used to generate the first useful

word for the output (in original training it was used the token "<|startoftext|>").
In order to understand what kind of text GPT-2 produces and its limits is useful to take a look at the dataset on which the network was pretrained.
A. Radford et.al. constructed a very large dataset in order to obtain samples the most "general purpose" as possible. This was done by exploiting web scrapers (bots able to extract HTML from websites through browsers) to retrieve "well done" text from a social media platform (Reddit). The resulting dataset, called "WebText", contains texts coming from over 45 million links.

3.2 Data available and retrieved data structures

First step of the case of study is to retrieve data that could be useful to train the network and construct useful data structures so a description of available files and what has been extracted from them is needed.
One of the first useful file contains the list of all European top competition matches from 2000 to 2020 with results, teams and various statistics on each game.

teamId	MatchId	match	team	teamCountry
teamId1	matchId1	team1 v team2	team1	team1Country
teamId2	matchId1	team1 v team2	team2	team2Country
teamId3	matchId2	team3 v team4	team3	team3Country
teamId4	matchId2	team3 v team4	team4	team4Country
teamId5	matchId3	team5 v team6	team5	team5Country

Table 3.1. example of one page belonging to matches file, here it is not shown dates and stats that were available in the original file

The information are stored as a csv file with multiple pages with the format shown in table 3.1: each pair of records describes a match between two European teams and each record belongs to a pair and gives information about one of the two teams like country to which the team belongs and stats retrieved during the match played by the team.
The file is used to retrieve two important information: by reading each page two data structures are created, the first one is a python dictionary useful to retrieve the name of a team given its unique id and the second one is a

Python list containing the ids of all the matches. Both data structures are then stored in JSON files to simplify their future use.
Another important file to retrieve useful information is the one containing the list of players belonging to teams that from 2013 to 2020 took part in elite European competition.
Like in matches file case, players are stored in a multi-page csv file with their ids, name, belonging team and other important stats as number of goals and assists performed during the competition, sum of played minutes etc. as shown in table 3.2

playerId	playerName	teamName	matchesPlayed	goalsScored
playerId1	playerName1	teamName	matchesPlayed	goalsScored
playerId2	playerName2	teamName	matchesPlayed	goalsScored
playerId3	playerName3	teamName	matchesPlayed	goalsScored
playerId4	playerName4	teamName	matchesPlayed	goalsScored
playerId5	playerName5	teamName	matchesPlayed	goalsScored

Table 3.2. example of one page belonging to players file, here it is not shown other stats as assists and minutes played that were available in the original file

To retrieve a reliable data structure from this file a little attention has to paid: a player belonging to a team can be sold or lend to another one once the competition in a year finishes or before it starts, so it is not possible to create data structures based on teams. From the file described above is retrieved a Python dictionary that giving the id of a player returns the real name, then the data structure is stored in a JSON file for future use.

teamName	coatchId	coatchName	matches
teamName	coachId1	coachName1	matchesPlayed
teamName	coachId2	coachName2	matchesPlayed
teamName	coachId3	coachName3	matchesPlayed
teamName	coachId4	coachName4	matchesPlayed
teamName	coachId5	coachName5	matchesPlayed

Table 3.3. example of one page belonging to coaches file

Similar file to the one described above is the one giving information about

coaches (showed in table 3.3): as before, information are stored in a multi-page csv file with a record for each coach containing useful information such as coach name and id, team in which that coach worked in a specific year and number of official matches where they appeared. Taking into account the problem described in players file, also in this case a Python dictionary with the pair coachId, coachName is created and stored.
some of the most important files from which useful information can be obtained are marks: in the world of sport, a mark represents an event that occurred in a certain minute in a specific game (e.g: goal, substitution, yellow/red card but also less important occurrence as shots, off-side etc.).

```
[{markId1: Id;
Tags:["Goal"];
OfficialTime:time;
MatchTime: minute of the match;
BodyPart: part of the body used to score
Subjects:[
{Verb: subject who scored or suffered the goal
 Type:Team
 Id:teamID
}
{Verb: subject who score or suffered the goal
 Type:Player
 Id:playerID}
.
.
.
]
}
 {markId2: Id;
 Tags:["YellowCard", "RedCard"];
OfficialTime:time;
MatchTime: minute of the match;
Subjects:[
{Verb: Received
 Type:Team
 Id:teamID
}
{Verb: Received
 Type:Player
```

```
  Id : playerID}
  }
    .
    .
    .
   {markIdn:  Id
    .
    .
    .
}
]
```

So, once the structure of this kind of files is clear, a lot of useful information can be retrieved. A marks file is stored as a JSON structure containing several Python dictionaries (marks) and each mark has different keys and values depending on which type of event it describes.
Generic file structure and Two example of marks are shown in the simplified JSON representation above. The first mark is an example of a "Goal" event, organised as a dictionary with a key for each relevant information as minute of the match when event happened and part of the body used to score and values that can be simple strings, lists (like in the case of the key "Tags" where the type of event the mark is describing can be retrieved) or another dictionary (like in the case of the subjects involved in the event, as it can be seen looking at the example of file: in "Goal" marks this dictionary contains the id of the team and the player that scored but also which team and player, most of the time the goalkeeper, that suffered a goal).
The second mark is a simplified example of a "Booking" event (card received from a player). In this case the structure of the dictionary is quite the same but few differences can be noticed: besides usual information such as minute of the game and tags (that in this case discerns if the specified player received a yellow card, a red card or both) in the case of "Subjects" dictionary only the team and the player that were punished is stored.
The starting training set on which GPT-2 is trained is retrieved from a text file containing sentences belonging to existing articles with meta data for each text where information as type of article (match background, match report etc.) and ids of match and teams involved are stored. Finally, texts coming from more than 25700 articles are saved and used as starting dataset.

3.3 First approach: let GPT-2 work freely

The simplest approach to get first results and check how the network behaves is to train the network with texts described at the end of section 3.2.
For this purpose, the model used is retrieved from "Hugging face"[1], the most important and used library concerning natural language processing and transformers.
The library allows to create and train different models of the network, from the smallest to the biggest one, with different configurations and different frameworks (Tensorflow, pyTorch). As first step, a Tensorflow model pretrained on "Webtext" dataset (described in section 3.1) with default parameter is used in order to understand how much the network can fit the starting training set.

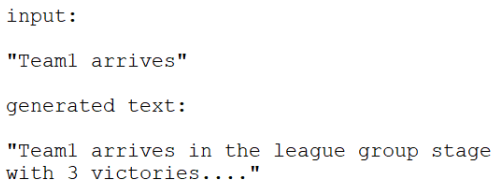


Figure 3.1. example of text generated by GPT-2 trained on articles

The figure 3.1 shows an example of text generated by the GPT-2 model trained on matches articles. The decoding strategy used in this case is a beam search (with number of beams = 7) with the temperature value of the softmax layer set to 0.9.
It is noticeable that, even if raw text regarding soccer is given as input to the network, the model is already able to create semantically and grammatically good sentences.

3.4 GPT-2 training based on keywords

As it can be easily guessed looking at the generation example in section 3.3, the main problem of a simplified approach is that GPT-2 learns in a very good way how to construct a well done text both grammatically and semantically but there is no way to guide the network for understanding the context

[1]description at: https://huggingface.co/gpt2

in which sentences are created.
A good way to overcome this issue would be to give to the model a sequence of keywords that could drive the output text both in training and generative phase.
Since, as seen in section 3.1, GPT-2 generates text by predicting next words given an input text, a good way to go could be to give as input text for the network several keywords in order to lead the model to create sentences that follow the context given by the structured input.
First attempt regarding this approach is made by extracting the useful input words directly from input text right before the training phase.
As first step a set of keywords is created by exploiting created data structures: all the names of teams and players that ever played at least one match in the official history of the elite European competition are retrieved from dictionaries described in section 3.2, then other specific keywords are added such as all the possible phases of the competition (final, semi-final, quarter-final etc.) and all the possible results (from 0-0 to 12-12).
Besides keywords, another important issue that affects results is the divergence of texts: articles available are not focused only on description of a specific match but also on the background and history of the teams (how many times the clubs met and with which results, how well they did in previous matches, how good important players did in previous games etc.), information that could confuse the network and lead it to loose the context and useful sentences in the training set. Since it is not possible (for time and human resources reasons) to read each article choosing useful sentences, an empirical approach is performed by retrieving only first sentences of texts (those who usually contain more information regarding the match to which the article refers).

```
text:

"They then followed up with a dramatic comeback victory against
away_team in the quarter_finals, late goals from home_player1
and home_player2 securing a 2-1 win at the home_team_stadium"

keywords:

"home_player1,2-1,home_player2,quarter-
final,away_team,home_team,final"
```

Figure 3.2. representation of one record with simple keywords generation

The figure 3.2 shows one sample retrieved from articles and the associated keywords.
During training, keywords are put as first words followed then by the actual text. As it can be easily guessed, this first kind of approach is likely to fail

for several reasons:

- the order of keywords types changes in different samples (e.g. goals are not always followed by scorers)
- textual search of keywords lead to miss important information because of the presence of semantic synonyms (as the pronoun 'They' in the example)
- this kind of search lead also to add keywords actually not belonging to text (in the example, the extracted keyword 'home_team' is actually not referring to the team but to the name of the stadium in which the match is played).

Figure 3.3. example of text generated by GPT-2 trained with keywords extracted directly from text

```
input:

"home_team, home_player1, 2-0, away_team, semi-final"

generated_text:

"With a 3-2 victory against third_team, home_team reaches semi-
final for the third time in five years...."
```

The issues explained above can be noticed in the text generated by a "top k" decoding strategy, with k set to 20 and temperature set to 0.7 (figure 3.3): even if some of the input keywords can be found in the text, the network continues to not following text given as input (e.g. it writes a 3-2 result rather than the one given as keyword) and also adding useless information (e.g the home team reached semi-finals for the third time in five years) not obtainable from starting words.
So a new approach has to be found in order to produce a more structured set of keywords related to the input text but not extracted directly from it. Following the lead of Max Woolf [2] a new "Encoder" class is created with the task to produce a set of keywords with a well defined structure.
Given a text and a set of keywords the class tries to adjust the order of the words with the aim to give to the network an idea of the scope and this is done by creating an input sample composed of different parts: "category" where keywords giving the context of the input text are put, "keywords" where meaningful words appearing or guiding the text have to be written and "Title" representing the actual input text in samples.

[2]link available at: https://github.com/minimaxir/gpt-2-keyword-generation

Unfortunately there is no mathematical or theoretical reason behind this use of keywords besides the attempt to debias the input text for the network, however, empirically it can be noticed that, when a structured and hierarchical representation of significant words in the sentences is used, GPT-2 seems to approach better what is expected in a controlled text generation task.
Since the keywords have to appear in the final input samples as the first part of the text, characters not belonging to original text have to be used to distinguish both keywords and text and different functions (category/actual keywords) of tokens. After a careful research in the text (searching for characters not appearing in original sentences) some special characters are chosen as delimiters: the special character "‘" indicates the "section" keyword, " ∧ " represents the start of the keywords belonging to input text, "@" indicates the starting point of text to be generated and the special character " ∼ " represents the starting point of each part of input (section, keywords or text).
In order to create desired input with a fixed structure for each article, keywords need to be extracted in a way such that a well understandable scheme is followed. To do so the marks described in section 3.2 are used within new kind of files.
First, words driving the input have to be chosen and looking at sentences it is clear that information giving meaning to articles are entities like phase of the competition (final, semi-final etc.), name of teams, result of the match, name of players who scored or suffered goals, name of coaches.

```
for each match with available marks:
    keywords=get_keywords(match_marks)
    save_keywords_in_file(keywords)

get_keywords(marks_file){

home_team, away_team = get_teams_name()
home_coach, away_coach = get_coaches_name()
match_type = get_competition_phase()

home_goals=0
away_goals=0
home_scorers=list()
away_scorers=list()
home_suffers=list()
```

```
away_suffers=list()

for each mark containing tag "Goal":
    if mark contains tag "OwnGoal":
        assign_goal(home_team, away_team, OwnGoal=True)
        add_scorer(home_scorers, away_scorers, OwnGoal=True)
        add_suffer(home_suffers, away_suffers, OwnGoal=True)
    else:
        assign_goal(home_team, away_team, OwnGoal=False)
        add_scorer(home_scorers, away_scorers, OwnGoal=False)
        add_suffer(home_suffers, away_suffers, OwnGoal=False)

keywords = list()
keywords.add(match_type, home_team, away_team, home_goals)
keywords.add(away_goals, home_scorers, away_scorers)
keywords.add(home_suffers, away_suffers, home_coach, away_coach)

return keywords
}
```

The pseudo-code above represents the script used to extract keywords. For each events file belonging to specific articles, keywords are retrieved by exploiting information contained in mark describing goals paying attention for the special case of own goals, in particular: first keywords returned are the type of match, teams and coaches name, then two integer numbers representing goals scored by the home team and the away one, finally, four lists representing name of players who scored or suffer the goal for the respective team. Since the marks don't contain names of entities but their ids,the use of data structures to map id of entities in name described in section 3.2 is essential.
Once keywords for each top European competition match are stored, they are given as input to the encoder class within the original text belonging to the respective article.

Figure 3.4 shows a final training input sample given to GPT-2: the type of match is given to the network as "section" keyword while the other tokens previously described are used as list of keywords belonging to text (the term "belonging" has not to be interpreted as "present in text" but as "words that could be or not written in the original article but in any case give meaning

```
<|startoftext|>~`group~^home_team away_team 2 1 {'home_player1'
'home_player2'} {'away_player1'} {'away_goalkeeper'
'away_goalkeeper'} {'home_goalkeeper'} home_coach
away_coach~@home_team fight back to beat away_team at stadium_name
away_coach gives away_team fourth-minute lead  but home_player1
scored...
```

Figure 3.4. representation of one sample with structured keywords generation

to the text").
Figure 3.5 shows an example of text generated by GPT-2 trained with structured keywords with a "top k" decoding strategy, with k set to 20 and temperature set to 0.7.

```
input:

"<|startoftext|>~`semi-final~^home_team away_team 2 1
{'home_scorer1'  'home_scorer2'} {'away_scorer1'} {'away_sufferl',
'away_suffer2'} {'home_suffer1'} home_coach away_coach~@"

generated text:

"Home_team's dream European league journey goes on after they beat
away_team to book a place in the last 16 as group winners with a
game to spare thanks to home_scorer1 and home_scorer2...."
```

Figure 3.5. example of text generated by GPT-2 trained with keywords extracted from marks

It appears clear that the way entities are now given as input lead the network to generate a text more strictly related to words given as start. However the issues previous explained remain unresolved since the network continues to add information not related to keywords to generated text.

3.5 Why GPT-2 leads to poor results

Looking at the results obtained with GPT-2, even without evaluating models, it seems clear that the network is not following very much the attempts to guide the text generation.
The model seems to start basing the generation on the given keywords but then it diverges to create text always correlated to the world of soccer but without any acknowledge of the context and the way traced before.
In the official Github repository of GPT-2 model[3] in section "model_card"

[3]link available at: https://github.com/openai/gpt-2

it is explained which use case GPT-2 is intended to achieve:

"Because large-scale language models like GPT-2 do not distinguish fact from fiction, we don't support use-cases that require the generated text to be true.
Additionally, language models like GPT-2 reflect the biases inherent to the systems they were trained on, so we do not recommend that they be deployed into systems that interact with humans unless the deployers first carry out a study of biases relevant to the intended use-case."

So reading what the authors of the model wrote, it appears clear that forcing GPT-2 to follow some sort of schema based on real facts is a not so easy task for two main reasons: The network itself was not build with the will to base its generation on real facts but only to create text from scratch without predefined directions and moreover GPT-2 is inevitably related to the dataset it was trained on which does not consider the interaction (directly or not as question and answer tasks or guided text generation using keywords) with humans.
Since there is not the availability of a large (millions of samples) training set about the task this use case is about to complete, a brand new way has to be taken.

Chapter 4

T5 and Data-to-Text Generation

4.1 Data-to-text Generation

Since the task of guided generation of text from scratch seems not a viable road with the available amount of data, the case of study switches to a new kind of text generation: rather than expecting a large text from a set of words given in input, a better way to guide the generation could be to start from a table of data containing the stats about the match expecting as output sentences describing each record of the input structured information.
The new approach followed is called Data-to-text Generation and it is defined as the task of generating text from a data source.
In order to perform data-to-text generation, first a new architecture able and built to accomplish this kind of task has to be used, a dataset has to be constructed in the way the new model requires and, working now with tabular data with a well-defined structure, different evaluation metrics have to be used to check results.

4.2 T5: operations and original training

As explained in section 2.2, T5 is a large transformer-based architecture trained on a huge unlabeled dataset (C4) to achieve in the best way multi-task learning. The authors of the neural network intended and trained the model to be fine-tune in a very big range of tasks by treating each problem as a text-to-text work: during training, the model tries to produce new text,

even for tasks usually not requiring this kind of output (as classification and regression problems). Even if there is no theoretic evidence that this kind of approach should work better than others, T5 reached terrific results especially on natural language processing tasks.
The way in which T5 obtains an output text from input is very similar to the way original transformer works:
The input sequence is first of all embedded and given to the first encoder that processes the incoming vector to a self-attention layer and feed forward network producing an output that is given to the next encoder and so on until the last encoder block is reached, then the last encoded output is transformed into attention vectors of keys and values that take into account the relation between input tokens. The vectors described above are given as input to the encoder-decoder attention layer of each decoder helping it to focus only on appropriate tokens of the input sequence. Each decoder, besides the specific component used to exploit the knowledge retrieved from encoders, has a structure very similar to the encoding one so also in this phase each output is given to the next block until the final decoder is reached whose results are processed by a fully connected (linear) and a softmax layer to obtain words with respective probabilities.
As said, T5 has been trained first on a huge unlabeled text set in order to achieve then good performance on different tasks. A closer look to starting dataset (C4) is needed to obtain a good knowledge about what the network can do and how new data can be extracted to construct new datasets.
The Colossal Clean Crawled Corpus (C4) was built by exploiting Common Crawl as source of text scraped from web. Common Crawl is a publicly available archive providing web extracted text with no markup and other non-text content from scraped HTML files. Since the original text contained not human text (e.g. HTML source code, error messages) and also natural text not useful for NLP tasks (e.g. offensive language, shopping lists) a very careful clean up has been made:

- kept only text ending with termination punctual mark (e.g. period, question mark).
- discarded pages containing only fewer than 5 sentences and lines with less than 3 words.
- deleted pages containing obscene or offensive language
- discarded pages containing the word "Javascript" and the curly bracket ("{") in order to eliminate pages containing code or error messages (in

natural language usually this kind of words are not present)

- discarded also all the pages not written with English language text.

The resulting "C4" text is a huge dataset composed by 750 GB unlabeled data of quite clean and natural English text about various topics.

4.3 First approach: from GPT-2 to T5

The obvious first step to check how much better T5 can rely on an input sequence to generate text than GPT-2 is to use the same training set described in section 3.4 comparing the output sentences retrieved by the two networks. First of all, the dataset used with GPT-2 has to be transformed in a structured input more compliant to what T5 expects.
The base Tensorflow model of the "hugging face"[1] library used in this case of study requires in training phase an input composed of three well-defined parts: "target_text", the ideal output text that the network should generate, "input_text", input sequence to be processed, "prefix", string representing which task has to be accomplished (in this case the string "create_text" is used).

```
input_text:
"['home_team', 'away_team', '2', '2', ""{'home_scorer1'
'home_scorer1'}"",
""{'away_scorer1'  'away_scorer2'}"", ""{'away_goalkeeper'
'away_goalkeeper'}"",
""{'home_goalkeeper'  'home_goalkeeper'}"", 'home_coach',
'away_coach']"

target_text:

"away_team will feel aggrieved to exit the league after
coming so close, but at this stage of the competition, against
opponents like home_team, the smallest of details can prove
decisive. home_scorer1's goals lead home_team to pass to the next
phase even after away_scorer2 scored..."

prefix:
create_text
```

Figure 4.1. example of the first structure of training samples for T5

In figure 4.1 it is shown how keywords and training samples in general are transformed to be compliant to a T5 input.

[1]description available at: https://simpletransformers.ai/docs/t5-model/

Figure 4.2. first text generated by T5 on a fictional input sample

```
input_text:

"['home_team', 'away_team', '2', '2', ""{'home_scorer1'
'home_scorer1'}"",""{'away_scorer1' 'away_scorer2'}"",
""{'away_goalkeeper' 'away_goalkeeper'}"", ""{'home_goalkeeper'
'home_goalkeeper'}"", 'home_coach', 'away_coach']"

generated text:

"home_team mounted a remarkable comeback to deservedly draw at
away_team. The home side flew out of the blocks with home_scorer1
on target twice inside nine minutes..."
```

the first text generated by T5 from a fictional input sample is shown in figure 4.2, as decoding strategy a "top k", with k set to 20, and a "top p" approach, with p set to 0.95 are combined. It is noticeable that, although an input not exactly compliant to the structures T5 expects is used, the model seems to follow given keywords in a same or even better way than GPT-2.

4.4 Totto dataset and use in T5

Once understood that an approach of text generation based on input structured could lead to a better results rather than trying to guide a pure text creation, a way to structure data in a compliant way to what T5 expects has to been found. Looking at benchmarks on data-to-set experiments the dataset closer and more similar to what the case of study means to do seems to be the one published in 2020 by Ankur P. Parikh et al.[7] whose description could be helpful to understand why certain decisions in the dataset self-built for the use case need to be taken.
Since data-to-text generation is defined as the task of generating a target textual description y conditioned on source content x in the form of structured data, a form of input can easily be represented by tables.
Totto is an open-domain English table-to-text dataset with over 120,000 training examples composed by Wikipedia tables with enlightened cells and target sentences for each related table. In order to construct the state of the art dataset for data-to-text generation several steps were needed:

- To obtain only Wikipedia pages containing tables-sentences pairs describing statistics (sports, politics etc.), retrieved only tables and sentences having in common at least three non-date number with non-zero digits.

- To retrieve only sentences related to tables, kept only texts with at least 3 distinct cell contents from the same row in a table
- Annotate tables containing links that drove to other pages containing sentences related to tables of the previous pages

After these automatic steps a more human based clean up phase was performed by "annotators":

- Table readability: deletion of poorly formatted or not understandable tables.
- Cell Highlighting: highlighting of cells that supported a phrase considering a sentence supported by a table when part of it was present in cells content or in table's metadata or when the analyzed text could be logically inferred by the table.
- Phrase deletion: removal of sentences unsupported by the highlighted cells
- Decontextualization: modification of sentences containing ambiguous pronouns and nouns with respective tables entities.
- Secondary Annotation Task: correction of grammatical errors possibly still present in texts after previous steps.

The final dataset retrieved [2] contains more than 120.000 samples with over 1.200.000 overall target sentences.

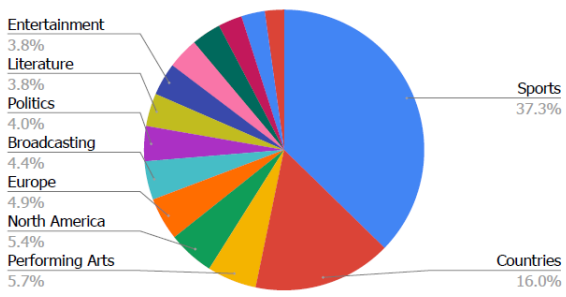


Figure 4.3. distribution of different topics in Totto dataset

[2] publicly available at: https://github.com/google-research-datasets/totto

In figure 4.3 the distribution of topics in the final obtained Totto dataset are shown, as it can be seen a large number of tables and sentences generated regards sport (about 45.000 samples).
After having described how a well-done data-to-text dataset can to be structured (tables with record containing entities and metadata and short sentences strictly related to important cells), the way this kind of structure could be given as input to T5 has to be found.
In 2020 Mihir Kale in his paper[8] studied the "pre-train + fine-tune" strategy for data-to-text tasks comparing results obtained from old state of the art model (like Bert) with T5 on three different dataset:

- "WebNLG" to convert graphs in textual description.
- "MultiWoz" to convert slot of key-value pairs in textual question and answer.
- "ToTTo" to convert tables with highlighted cells in text describing records.

T5 seems to outperform in each benchmark the previous state of the art but what is important for this case study is to understand how the tables belonging to ToTTo are transformed in order to be accepted as input from the transformer-based architecture.

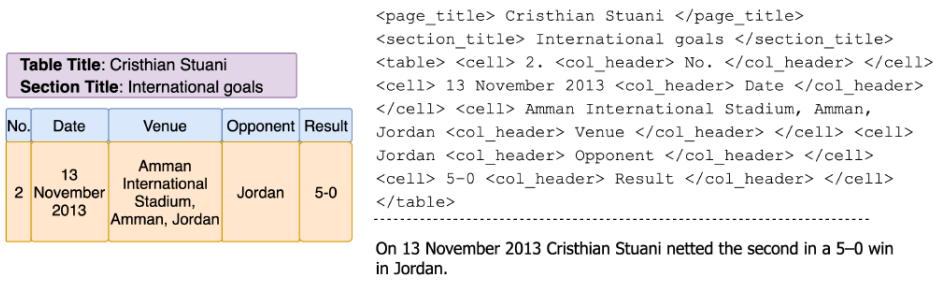


Figure 4.4. example of how ToTTo records are linearized to be used in T5 fine-tune

As figure 4.4 shows, the transformation of ToTTo samples to be compliant to T5 input structure is quite simple: all the pages belonging to the dataset are linearized using a HTML-like format in which tags represents entities of the sample.

- The title of the page and each section have a specific tag with the "page_title" tag in common for all the sections and tables belonging to that specific page.

- each table belonging to a specific section is indicated by the tag "table".
- Each cell containing a value for a specific entity is started with the tag "cell" and the header of that cell is positioned right after the actual value in the tag "col_header".
- the target sentence is left as is.

The way ToTTO records are transformed appears quite reasonably: since T5 is built as a text-to-text neural network every kind of data that has to be processed need to be first turned into a text-like input, in this way the network is able to treat each problem as a translation task that, as seen in section 1.3, can be easily solved by a transformer-based architecture.

4.5 Data retrieved

Once understood how data have to be structured in a data-to-text problem and how to transform them to be used as input in T5, it appears clear to have a big issue regarding data explained in section 3.2: as seen, to make T5 work properly, texts used as target sentences have not to be too much long and also they have to be very correlated to the metadata given in input.
Data retrieved from top European league is made of too much tokens and also it is too divergent (texts contain, besides sentences strictly related to the analyzed match, also information about the history of teams and about key events occurred in previous face to face matches).
Since target texts available were not useful for training the network, two possible approaches could be chosen:

1. Manually modification of existing text in short sentences strictly related to events described by metadata.
2. Creation of new target sentences obtained from different sources.

Since the first approach is not possible for human resources and time reasons, new sources of text have to be found.
As seen in section 4.4 and in all the dataset previously described, the best way to obtain text of good quality is to use web scraper to get text from HTML pages regarding the interested topic.
Chosen this kind of approach new issues arise regarding web pages and text obtained:

- how difficult is to make a connection between already available metadata and web pages in order to navigate on links that have text actually related to the match stats are referring to.
- how hard is to obtain text strictly related only to the events described on match stats.
- how difficult is to make a connection between each stat of a specific match and each sentence retrieved from web for that match.

Which are the first websites from where to extract texts describing match are very easy to choose: The official website of the elite European league is a good starting point for several reasons:

- each URL has a base string in common for each match and its last part is composed by an id exactly corresponding to the id of marks file from where metadata are retrieved.
- Each web page of the official site has a list of minute-per-minute events happened during the match with comments.

Unfortunately, the description of events in pages has not a well-defined structure or a title from which it could be understood what is described (a textual research for entities in texts could be a solution but as seen in section 3.4 this approach probably lead to very bad results, considering also that for only one minute more than one commented text not relating to the event is written) so as starting point only final comments and description of final verdict are taken.
In order to extract HTML text from web pages two Python libraries are used: the first one, "BeautifulSoup4"[3] allows to scrape HTML text from web, navigate through tags and extract text, the second web scraper, "Selenium"[4] allows to interact with pages before extracting information (e.g: scroll until the bottom to make the page load new information, click buttons etc.).

```
for each match_id:
    url = base_URL + match_id
    page = get_URL(url)
```

[3]documentation available at: https://www.crummy.com/software/BeautifulSoup/

[4]documentation available at: https://www.selenium.dev/

```
page.scroll_until_interesting_text_found()

for each text_with_tag_p:

    if text.contains("Full-time-verdict"):
        write text into "match_id.txt"

    if text.contains("final-comment"):
        write text into "match_id.txt"
```

The pseudo-code above shows how the created script works: for each id in the list of matches with available metadata, first the page object to interact with the web site is gotten, then the page is scrolled until the text containing what has to be taken is reached, finally, since the interesting text is located in p tags in the HTML document, in the file named with the id of the current match it is written the text present in each "p" tag that contains the title searched.
Once texts are available, in order to construct the data set is worth only to create a table with metadata as cells and sentences coming through the text file whose name contains the same id of the the mark file from which stats are extracted as targets.

Since the number of sentences obtained from the official website of the top European competition are not so numerous (for each match id up to two lines of resulting text are retrieved bringing the number of targets up to 260) a new source of text is needed.
A good description of important events in European football matches is contained in the sport section of the "BBC" website (the main British public service broadcaster). As in the previous case, the URL that brings to a web page is composed by a base path plus a unique id used to indicate a specific match. What is interesting is that in BBC pages a "Live text" section describes minute per minute what happened during the game and most important events (yellow/red cards, goals and substitution) are highlighted in the HTML[5]. Using web pages not coming from the same source of metadata lead to different problems but also to advantages:

- the match ids of marks and BBC URL are not the same so there is no direct correspondence to exploit.

[5]example of a match web page available at: https://www.bbc.com/sport/football/43621073

- the minutes in which an event occurs can be different between the official website of the league and the British broadcaster (there can be a difference up to a value of one because of the moment on which the event is annotated during the match)
- Since important events are highlighted, they can be easily retrieved from the HTML text.
- some of the entities values with special characters could be simplified (e.g: removing accents in names) in one source and not in another.

For what concerns ids correspondence, since a list of BBC ids with name of the teams is not available, it has to be done manually checking also for each web page if the list of events is available (this lead to eliminate about four sample of the original list of ids).

```
for each BBC_id:
    url = base_URL + match_id
    page = get_URL(url)
    live_text_button = page.find_button("Live-Text")
    live_text_button.click()
    text_available = True
    while(text_available):
        for each highlighted_text:
            write text into "BBC_match_id.txt"
        try:
            next_events_button = page.find_button("Next-page")
            next_events_button.click()

        except:
            text_available = False
```

The pseudo-code showed above describes how the script for BBC events works: unlike official league website, in this case the events are not reachable directly after having load the page but a little interaction with the website is needed. First, gotten a web page corresponding to a match, the button "Live text" has to be pushed to access events occurred during the game (button pressure is done by Selenium functions), then not all the events are already available so, after having retrieved the highlighted text contained in the page, the button "Next page" has to be clicked in order to make Ajax load other

information from the web server. The extraction of sentences for each id stops when no "Next page" button is found so no remaining pages available are found.
The use of a third part text increases significantly the amount of samples: since up to 15 sentences are retrieved for each match the total number of sentences is now made of about 1700 lines.

```
for each id in official_id:
    BBC_id = ids_correspondance(id)

    for each event in BBC_id:
        if event in official_events:
            if event[minute] in set(official_event_minutes):

                if event=='Goal':
                    build_record()

                if event=='Card':
                    transform_text(BBC_text, official_stats)
                    if(BBC_team in official_mark) and
                    if (BBC_player in official_mark):
                    build_record()

                if event=='Substitution':
                    transform_text(BBC_text, official_mark)
                    if(leaving_player in official_mark) or
                    if (entering_player in official_mark):
                    build_record()
```

Once the text sample are ready, there is the need to construct records linking events from official stats with the ones belonging to web pages. The pseudo-code above shows how the link is made: for each official match the BBC id is found (exploiting the correspondence data structure manually built) then, for each match, events are linked by comparing events in a short amount of time (one minute before or after the official time or in the exact time). Finally, for each type of events a different action is performed:

- "goal" events are stored as is (making as assumption the impossibility that 2 goal can be scored in the same minute)

- "card" events are stored only if the name of the team and the player match in official stats and BBC text (to avoid the possibility of wrong matches coming from punishment of more than one player in a time frame of 2 minutes)
- "substitution" are stored only if the name of at least one player matches (to avoid the case of more than one substitution in the same time frame)

In order to overcome the unfortunate case in which name of players and teams are simplified in one of the two sources a transformation is applied before comparing: both stats and texts are flatted (accents and special characters are eliminated) and all the characters are transformed in lower case to obtain matches.
Once texts have been extract and before building the input samples for the network, the way to proceed has to be chosen. Two main approaches can be followed:

1. Extraction from ToTTo dataset of the samples with topic "sport" and linearization of European league metadata following the exact rules of tables linearization by Mihir Kale, performing then a double fine-tuning by first giving to T5 sport samples of Totto and in a second phase data created from matches.
2. Taking as example ToTTo linearization, find a way to linearize metadata performing then a single fine-tuning

As seen in section 4.4, the number of ToTTo sport tables retrieved is about 45000 while the available tables for this case study are close to 260 in the first data set and to 1700 in the second one: performing a double fine-tuning could probably lead to a biased model too much influenced by the bigger number of samples and too little aware of interesting samples so the second approach is clearly the most reliable one.

Figure 4.5 shows one sample belonging to the training set built with the league's official website: what is remarkable is that the input text (stats) is given to the network as a structure very similar to how the linearization of ToTTo's samples works: each input text has a pseudo-HTML structure with a tag for the title and one for each cell with the respective end tag, what is different than what has been seen in section 4.4 is that in this case of study each target text belongs to one record so there is no need to group records in a table (one record corresponds to one sample).

```
target_text:
"home_team reaches the last eight of the league thanks to
strikes from home_player1, home_player2 and home_player3. The
away_coach team exits the competition at this stage for the
first time, but they did not defend well enough over the two
legs and it proved costly for away_team."

input_text:
<title>home_team 3-1 away_team</title><home_scorers>
{'home_player1'  'home_player2'  'home_player3'}</home_scorers>
<away_scorers>{'away_player1'}</away_scorers><home_suffers>
{'away_goalkeeper'  'away_goalkeeper'  'away_goalkeeper'}
</home_suffers><away_suffers>{'home_goalkeeper'}</away_suffers>
<home_coatch>home_coach</home_coatch><away_coach>
away_coach</away_coach>

prefix:
create_text
```

Figure 4.5. example of one input retrieved from official website

```
target_text:
"Goal! home_team 3, away_team 1. home_player1 (home_team) right
footed shot from the centre of the box to the high centre of the
goal. Assisted by home_player2."

input_text:
<title>home_team vs away_team</title><event>Goal</event>
<minute>74'</minute><team>home_team</team><scorer>
home_player1</scorer><assist>home_player2</assist>
<result>3-1</result><body_part>Right</body_part>
<position>-0.853524,-0.853524</position>

prefix:
create_text
```

Figure 4.6. example of one input retrieved from BBC website

Figure 4.6 shows one sample belonging to the training set built with BBC web pages. looking at the sample belonging to the official website several differences can be noticed:

1. In BBC case, target texts are shorter and describe specific events occurred during the match.

2. Since new target sentences describe specific events also stats are made of more specific cells: in the example above it can be seen that in "Goal" occurrence not only the scorer is given as input but also other information as the position of the shot and the body part used to score.

3. As expected, name with special characters can be different in different sources (not visible in the example) but with the transformation performed during data set construction the match is found anyway.

Once data sets are ready they are shuffled and divided in training sets (85% of the number of samples) to be given as input to the model and validation ones (15%) to be used to check the goodness of final models.

4.6 Optimizers: AdamW vs Adafactor

Before starting with training phase, the optimizers available for the network has to be explained since they can have a big impact on the final resulting network. The library used gives two different choices on the optimizer, AdamW or Adafactor.
In 2015, Diederik P. Kingma and Jimmy Lei Ba proposed in their paper[9] a new kind of optimizer, called Adam, able to combine several features of previous solutions.
This optimizer can be seen as an improvement of the stochastic gradient descent algorithm that combines the advantages of two algorithms based on SGD: Adagrad and RMSprop. Since Adagrad maintains a fixed "learning rate" hyper-parameter (improving performances in NLP compared to classic SGD) and RMSprop updates its "learning rate" hyper-parameter looking at how fast the magnitude of the gradients change during different updates (improving results on noisy training set), Adam tries to combine both features by basing the learning rates updates on two different moments.

Figure 4.7. Adam base algorithm for weights update

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

In figure 4.7 is shown the base algorithm through which Adam optimizer updates weights. Given a set of hyper parameters, at each step the optimizer calculate values of the two moments that will be used for weights update after avoiding batches to become biased through a normalization made with an hyper-parameter for each moment (β_1, β_2). Such as SGD+momentum updates his learning rate so does Adam but with moments that are not fixed making the hyper parameter adapt in different phases.
Adam was first the most used optimizer algorithm since Ashia C. Wilson et al.[10] demonstrated that for a large variety of problems the algorithm

performs worst than classic SGD+momentum optimizer.
In 2019, Ilya Loshchilov and Frank Hutter in their paper[11] increased Adam results (calling this modified algorithm AdamW) by simply using weight decay instead of classic L2 normalization to reduce over-fitting demonstrating also that while for standard stochastic gradient descent they appear as the same equation, in the case of adaptive gradient algorithms, such as Adam, they lead to different results.
In 2018, Noam Shazeer and Mitchell Stern[12] proposed a new optimization algorithm, called Adafactor, based on Adam: the goal was to reduce memory usage preserving the benefits of adaptivity discovered by Diederik P. Kingma and Jimmy Lei Ba.
As seen in figure 4.7, Adam maintains in memory all the parameter for each weight to estimate the second moment meaning that the amount of memory needed becomes equal to the number of parameter. Adafactor overcomes this issue by maintaining only the per-row and per-column sums and estimating the second moment with these sums, decreasing the amount of memory needed to store estimators for a nXm weight matrix from $\mathcal{O}(n * m)$ to $\mathcal{O}(n + m)$.

Inputs: initial point $X_0 \in \mathbb{R}^{n \times m}$, relative step sizes $\{\rho_t\}_{t=1}^{T}$, second moment decay $\{\hat{\beta}_{2t}\}_{t=1}^{T}$ such that $\hat{\beta}_{21} = 0$, regularization constants ϵ_1 and ϵ_2, clipping threshold d
for $t = 1$ **to** T **do**
  $\alpha_t = \max(\epsilon_2, \text{RMS}(X_{t-1}))\,\rho_t$
  $G_t = \nabla f_t(X_{t-1})$
  $R_t = \hat{\beta}_{2t} R_{t-1} + (1 - \hat{\beta}_{2t})(G_t^2 + \epsilon_1 1_n 1_m^\top) 1_m$
  $C_t = \hat{\beta}_{2t} C_{t-1} + (1 - \hat{\beta}_{2t}) 1_n^\top (G_t^2 + \epsilon_1 1_n 1_m^\top)$
  $\hat{V}_t = R_t C_t / 1_n^\top R_t$
  $U_t = G_t / \sqrt{\hat{V}_t}$
  $\hat{U}_t = U_t / \max(1, \text{RMS}(U_t)/d)$
  $X_t = X_{t-1} - \alpha_t \hat{U}_t$
end for

Figure 4.8. Adafactor base algorithm for weights update

As it can be seen in Adafactor base algorithm (figure 4.8), the term α used to update weights is now calculated through a relative step size term $\{\rho_t\}_{t=1}^{T}$ and then multiplied by the scale of a parameter matrix defined as the root-mean-square of its components:

$$\alpha_t = max(\epsilon_2, RMS(X_{t-1}))\rho_t \tag{4.1}$$

the α_t term is also low bounded by a small parameter ϵ_2 to avoid the value zero for initial parameters. In few words, Adafactor proposes to reach similar

results compared to Adam using a very less amount of memory.

4.7 Evaluation metrics and results

In order to train the network with the training set created from official league website, one 12 GB CUDA GPU is used for 20 epochs.

Optimizer	Training time	Final loss
AdamW	5m 19s	0.061
Adafactor	6m 03s	0.066

Table 4.1. loss and training time to process the training set created with official league website

As it can be seen in table 4.1, AdamW optimizer seems to be faster and able to reach lower losses (it does not mean in this case that the produced results are better), this can be explained by the fact that Adafactor, by building new matrices to reduce memory used, requires more time to construct matrices during updates.
The losses produced by T5 are calculated by "Cross entropy": it is a measure of the difference between two probability distribution very used in deep-learning. This way to calculate a loss is based on the concept of "entropy": declaring y_i the probability of the i^{th} outcome, the entropy of a sequence composed by n tokens can be performed as:

$$e = \sum_{i=0}^{n} y_i \log \frac{1}{y_i} \tag{4.2}$$

and represents the amount of "surprisal" trasmitted in all the outcomes. The term surprisal is used with statistical meaning:

$$s = \log \frac{1}{y_i} \tag{4.3}$$

and represents the amount of useful information that an outcome gives (less probable outcomes bring more information than likely ones). In the case in which there is the need to understand the difference between a probability

distribution p_i and an estimation of that distribution d_i, the entropy measure becomes cross-entropy:

$$c = \sum_{i=0}^{n} p_i \log \frac{1}{d_i} \tag{4.4}$$

Since both equations 4.2 and 4.4 take into account the probability distribution Y as benchmark and since the second equation represents the difference between that distribution and one of its estimations, it is quite simple to understand that cross-entropy will be always greater or equal to entropy (equal when $p_i = d_i$).
Since cross-entropy increases when the estimation becomes further from the real distribution, it can be used as a loss function to minimize in order to bring the predicted probability distribution closer to the actual one. However, for this specific case study a measure that takes into account only how similar two texts are can be confusing to the network, that's why good metric evaluations have to be used.

```
input sample:

"<title>home_team 2-1 away_team</title><home_scorers>
{'home_player1'  'home_player2'}</home_scorers><away_scorers>
{'away_player1'}</away_scorers><home_suffers>{'away_goalkeeper'
'away_goalkeeper'}</home_suffers><away_suffers>{'home_goalkeeper'}
</away_suffers><home_coatch>home_coach</home_coatch><away_coach>
away_coach</away_coach> create_text"

produced_text (AdamW):

"away_team's win means a match for home_team but there are some
positives for the players. home_player1's late winner, coupled
with home_player2's magnificent display, leaves the visitors in a
bad situations..."

produced_text (Adafactor):

"It was a tough match against away_team, where the 2-0 aggregate
win means they can move on to the next round at the top of the
group."
```

Figure 4.9. text generated by the network trained on official comments

Figure 4.9 shows the generated text on two models trained on sentences coming from the official website with the optimizers described above given a fictional input cell, both texts are generated with a "top-k" plus "top-p" decoding strategy with $k = 50$ and $p = 0.90$. The issues described previously can be easily verified on the generated example:

- Because of the few number of samples, models are likely to overfit and add information not belonging or not obtainable from the input cell (it

can be noticed mainly on the result with Adafactor where information on previous matches and phases of the league probably belonging to training samples are present).

- Since the target texts in training samples are retrieved from final comments on the match (i.e. sentences not strictly related to what the cell describes) the output contains positive and negative comments not retrievable from cells (it can be noticed in sentences as "there are some positives for the players" and "It was a tough match").

Optimizer	Training time	Final loss
AdamW	1h 38m 04s	0.015
Adafactor	1h 46m 04s	0.016

Table 4.2. loss and training time to process the training set created with BBC website

The stats collected after training T5 on the dataset created using BBC web pages are shown in table 4.2, in this case the network is trained for 50 epochs. As expected, since the number of samples is bigger than the previous set, the average execution time needed to compute one epoch significantly increases and also the average loss reached appears clearly smaller because of the length and the tighter relation of sentences to the input metadata.

Figure 4.10. text generated by the network trained on BBC web pages (Goal)

```
input sample (Goal):

"<title>home_team vs away_team</title><event>Goal</event>
<minute>74'</minute><team>home_team</team>
<scorer>scorer</scorer><assist>assist_man</assist>
<result>3-1</result>
<body_part>Right</body_part>
<position>-0.853524,-0.853524</position>"

produced text (AdamW):

"Goal! home_team 3, away_team 1. scorer (home_team) right footed
shot from the
centre of the box to the bottom left corner. Assisted by
assist_man."
```

In figures 4.10, 4.11 and 4.12 are shown examples of generated text for three different events by the network trained on BBC web pages with a "top-k" plus "top-p" decoding strategy ($k = 50$, $p = 0.9$), results with Adafactor

```
input sample (Substitution):

"<title>home_team vs away_team</title>
<event>Substituion</event>
<minute>64'</minute><team>away_team</team>
<leave>leaving_player</leave>
<enter>entering_player</enter>"

produced text (AdamW):

"Substitution, away_team. entering_player
replaces leaving_player."
```

Figure 4.11. text generated by the network trained on BBC web pages (Substitution)

```
input sample (Card):

"<title>home_team vs away_team</title>
<event>Booking</event><minute>21'</minute>
<team>home_team</team>
<player>booked_player</player>
<yellow_card>yes</yellow_card><red_card>no</red_card>"

produced text (AdamW):

"booked_player (home_team) is shown the
yellow card for a bad foul."
```

Figure 4.12. text generated by the network trained on BBC web pages (Booking)

are not represented in this case because the generated sentences are exactly the same shown in figures. As said, since the target sentences are short and very specific also the generated text appears not so long as what has been produced in the previous case and also strictly related to input metadata.

The first metric evaluation used in this case study is BLEU, published by Kishore Papineni et al[13].
It is usually used as a metric for automatic machine translation evaluation in order to avoid human work that, besides the accuracy of the analysis, can requires months of work and is not repeatable with exactly same results. Bleu bases its metric on the assumption that "The closer a machine translation is to a professional human translation, the better it is." so the aim is evaluate how close automatic generated samples are to human generated sentences and to do so two main ingredients are needed:

1. a numerical “translation closeness” metric.

2. a corpus of good quality human reference translations.

This metric evaluation bases its scores on the accuracy metric and the first way to compare a generated sample with its reference could be to count the words shared and take this amount as quality of translation:

$$score = \frac{\#matches}{\#words\_in\_sentence} \tag{4.5}$$

However this kind of score could lead to favor long generated sentences that do not match a big number of words in the reference but simply contain a lot of words referring to the same token of the human text. This issue is solved in Bleu by the use of the so called "modified uni-gram precision":

- Count maximum number of times a word occurs in each reference text.
- Divide the total occurrences of each word by the count previously calculated.
- add the values found for each word and divide the final number by the total number of candidate words.

Figure 4.13. example of modified uni-gram precision of a text composed by one word

Candidate: the the the the the the the.

Reference 1: The cat is on the mat.

Reference 2: There is a cat on the mat.

Modified Unigram Precision = 2/7

In figure 4.13 it can be seen how with modified uni-gram precision calculation the score of matching goal decrease from 7/7 (standard precision) to 2/7.
As known, the meaning and context that a word could give in a sentence can completely change depending on which tokens precede or follow it, that's why words between candidates and references have to be compared also as n-gram (set of consecutive words).
Also in the case of n-gram the standard precision measure is not enough so the so called "modified n-gram precision" is used, a metric very similar to what has been described before but with the exception of using set of words rather than a single one.
Since standard Blue presents issues related to the phase in which the score is

retrieved (precision is calculated before decoding tokens) a modified version, called sacreBlue , created by Matt Post[14] of the evaluation metric is used: the score is now calculated each time after having decoded generated tokens.

Table 4.3 shows the different scores reached: as expected the precision reached with the training set containing BBC text results much higher than official comments, this is because texts belonging to the British broadcaster are much shorter and related to metadata than the ones coming from official web pages.

Dataset	Optimizer	Blue score
Official	AdamW	1.54
Official	Adafactor	0.42
BBC	AdamW	80.09
BBC	Adafactor	80.67

Table 4.3. Blue scores reached with described training sets and optimizers

The second evaluation metric proposed is PARENT[15] (Precision And Recall of Entailed Ngrams from the Table).
Blue metric bases its measures on the assumption that reference text is the perfect ideal output and all the targets have to be compared to it in order to get a reliable precision measure. However, in table-to-text generation, when data are generated automatically as in this case study, reference text could diverge from information contained in tables and not represent a good comparison basis. In order to calculate precision and recall, Parent uses respectively a union and an intersection of the reference texts and the tables to avoid not to take into account missing data in reference or consider incorrect information.
Each table T can be defined as a set of records composed of tuples (attribute, value):

$$T = \{r_k\}_{k=1}^{K} \tag{4.6}$$

By denoting G and R respectively the generated and the reference text that describe a specific table T, the evaluation set of tables, texts and reference generated by a model M can be represented as:

$$D_M = \{(T^i, R^i, G^i)\}_{i=1}^{N} \tag{4.7}$$

Since, as in the case of Bleu, there is the need to introduce more words to be compared in a single step, the set of n-grams of order (number of words in sequence) n in specific references and texts are indicated as R_n^i and G_n^i with $\#_{R_n^i}(g)$ and $\#_{G_n^i,R_n^i}(g)$ representing respectively the count of the specific n-gram g in R_n^i and the minimum between counts regarding the same n-gram on R_n^i and G_n^i.
In order to obtain precision and recall in this specific case study, since tables can not be directly compared to reference and generated texts, the notion of entailment probability has to be introduced and it represents how probable is that an n-gram in a text is correct given the associated table:

$$w(g) = Pr(g \Leftarrow T^i) \tag{4.8}$$

Parent's precision and recall take into account this entailment probability so the actual measures that the metric calculates are called "entailed precision" and "entailed recall".
In the case of precision, the task is to find the fraction of n-grams in G_n^i that are correct by considering an n-gram g "correct" if it is contained in the reference R_n^i or it has high probability to be inferred from the table T^i (so if the probability 4.8 is high). By denoting the probability that an n-gram belonging to G_n^i is also contained in R_n^i as:

$$Pr(g \in R_n^i) = \frac{\#_{G_n^i,R_n^i}(g)}{\#_{R_n^i}(g)} \tag{4.9}$$

the entailed precision can be calculated as:

$$E_p^n = \frac{\sum_{g \in G_n^i}[Pr(g \in R_n^i) + Pr(g \notin R_n^i)w(g)]\#G_n^i(g)}{\sum_{g \in G_n^i}\#G_n^i(g)} = \frac{\sum_{g \in G_n^i}\#G_n^i(g)w(g) + \#_{G_n^i,R_n^i}(g)[1 - w(g)]}{\sum_{g \in G_n^i}\#G_n^i(g)} \tag{4.10}$$

For what concerns recall of generated text, both the measure on the references ($E_r(R^i)$) and tables ($E_r(T^i)$) are calculated in order to take into account respectively sentences structure and information contained for each generation and then combined with geometric average:

$$E_r = E_r(R^i)^{(1-\lambda)} E_r(T^i)^{\lambda} \tag{4.11}$$

with the parameter λ used to decide the weight of each specific member. The average plays the role of a logic "and" representing so the intersection

between the two quantities.

$$E_r^n(R^i) = \frac{\sum_{g \in R_n^i} \#_{G_n^i, R_n^i}(g) w(g)}{\sum_{g \in R_n^i} \#_{R_n^i}(g) w(g)} \tag{4.12}$$

In the equation 4.12, representing the recall calculated on references, the term $w(g)$ (that is a contribution retrieved from tables) is used as a weight to exclude divergent texts (that will have low values of w).
Recall referring to tables, using the equation 4.6 and denoting $\bar{r_k}$ the string value of r_k, can be calculated as:

$$E_r(T^i) = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|\bar{r_k}|} LCS(\bar{r_k}, G^i) \tag{4.13}$$

with $\bar{r_k}$ number of tokens contained in the string value of a specific record r_k and $LCS(\bar{r_k}, G^i)$ the length of the longest subsequence in common between $\bar{r_k}$ and the generated sentence G^i (function used to ensure the same order of string entities between table and generated text).
Finally, entailed precision and recall can be described as two new measures able to estimate how good the generated text follows information retrieved from the union (precision) and the intersection (recall) of references and tables.
entailed precision and entailed recall are finally combined into an F_score measure to composed the actual Parent score for a single instance:

$$F\_score = \frac{2}{(\frac{1}{recall})(\frac{1}{precision})} = 2\frac{precision * recall}{precision + recall} \tag{4.14}$$

The final Parent score of an entire evaluation set for a model M is then calculated as the average Parent score of each instance:

$$PARENT(M) = \frac{1}{N} \sum_{i=1}^{N} PARENT(G^i, R^i, T^i) \tag{4.15}$$

Table 4.4 shows results obtained by applying Parent evaluation metric to available datasets and optimizers. Interesting information can be extracted from the table:

1. As in the case of Bleu, entailed precision of models obtained with BBC training set is much higher than what is reached with official sentences.

2. entailed recalls (and consequently F_scores) reach quite small value in both datasets.

The small values obtained with entailed recall can be explained by a different reason for each dataset:

- In the case of the training set obtained from official comments, the reference text is too much divergent compared to tables, this lead the intersection between the two measures explained in equation 4.11 to be very small obtaining as a result low values of entailed recall even if the generated text contains an acceptable number of table entities.
- In the case of the training set obtained from BBC sentences, even if generated text reaches high precision values, the length of all members of the equations (tables, generated and reference texts) lead to low upbounded values of recall.

Dataset	Optimizer	Parent precision	Parent recall	Parent F_core
Official	AdamW	0.36	0.03	0.04
Official	Adafactor	0.37	0.05	0.06
BBC	AdamW	0.93	0.08	0.12
BBC	Adafactor	0.94	0.08	0.12

Table 4.4. Parent scores reached with described training sets and optimizers

4.8 conclusions

Looking at the generated examples and results several conclusions can be inferred:

- The approach involving the use of T5 outperforms what is obtained with GPT-2, so a data-to-text approach in this case study seems to be the best way to proceed rather than the attempt to force pure text-generation networks to base their work on keywords.
- The length of sentences to be generated play a key role on the goodness of generation so a good trade-off has to be found: in the case of quite big amount of tokens (such as in the case of official comments) results seems to fall down but too much short sentences generation could result in a text not so correlated (sentences generated using BBC web pages describe only single events that need then to be put together to produce a

match summary and with no human interaction this lead to an accurate but not so fluent text).

- Besides the length, another important feature of texts to be in training phase has to be the low divergence: the training set has to be composed of sentences strictly related to and composed by entities belonging to tables (in the case of official comments data set, the long and divergent text leads to very bad results both in examples of generated text and in the evaluation metrics used).

Chapter 5

future works

5.1 GPT-3

Before abandoning generative pre-training networks an approach based on the new architecture created by openai, GPT-3,is worth to try.
Although the architecture and the specific feature of the model are not publicly available, what is known about the network is already very interesting for this case study:

1. The architecture of GPT-3 remains very similar to previous generative pre-training networks: the most important part of the model is still represented by multi-layer decoders as described in section 2.1.

2. The number of parameters of GPT-3 is increased up to 175 billions of parameters (the largest GPT-2 model reaches 1.5 billions) making GPT-3 the most powerful and largest language model ever created.

3. The main aim of generative pre-training is stress to its limit by training the network on about 500 billions of tokens in the unsupervised learning phase.

4. The number of parameters and the amount of data on which GPT-3 is trained leads the network to accomplish a wide range of tasks, including translation problems. So it can be used as a text-to-text translator like T5.

This features make GPT-3 very interesting for this case study since it combines the pros of both GPT-2 and T5 representing a hybrid network that

can solve data-to-text problems with the power of the best generative pre-training network.

5.2 Use of better datasets

As seen in this case study, besides the ability of the network to generate fluent text, another important factor in the goodness of results is represented by training sets having a good trade-off between divergence and length of sentences.
The first approach to retrieve better dataset is to find a data source (in official data available or through web scrapers) strictly related to matches events and not too divergent (e.g. not containing sentences regarding history of teams and players and descriptions of previous games in which they faced).
A second and more general approach could be to merge sentences belonging to the same data sources or coming from different founts.
As seen in section 4.5 the number of samples retrieved from BBC is much higher than the one obtained with official comments, unfortunately the sentences scraped are very short and describe specific events of the match. A way to obtain entire summaries strictly related to match events could be to manually merge sentences belonging to the same game, obtaining a number of training samples similar to the one obtained with official web pages (so quite low) and then increase the size of the training set by implementing one of the following approach:

1. Using the same data source (e.g BBC), obtaining more samples by scraping events for matches belonging to different soccer competitions, manually merging descriptions of important events and concatenating obtained summaries in order to build the final set.

2. Obtaining more samples by concatenating, to the existing set, samples coming from different sources (e.g different broadcasters with web pages describing events of matches) after having properly merged events obtained from a single match description in order to create a well-structured summary.

Bibliography

[1] Ashish Vaswani, Noam Shazeer, et al. *Attention Is All You Need.*

[2] A. Radford *Improving Language Understanding by Generative Pre-Training.*

[3] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever *Language Models are Unsupervised Multitask Learners.*

[4] Colin Raffel, Noam Shazeer et. al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.*

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer normalization.*

[6] Sennrich, R., Haddow, B., and Birch, A. *Neural Machine Translation of Rare Words with Subword Units.*

[7] Ankur P. Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, Dipanjan Das *ToTTo: A Controlled Table-To-Text Generation Dataset.*

[8] Mihir Kale *Text-to-Text Pre-Training for Data-to-Text Tasks.*

[9] Diederik P. Kingma, Jimmy Lei Ba *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION.*

[10] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht *The Marginal Value of Adaptive Gradient Methods in Machine Learning*

[11] Ilya Loshchilov, Frank Hutter *DECOUPLED WEIGHT DECAY REGULARIZATION*

[12] Noam Shazeer, Mitchell Stern *Adafactor: Adaptive Learning Rates with Sublinear Memory Cost*

[13] Kishore Papineni, Salim Roukos, Todd Ward, Wei-Jing Zhu *BLEU: a Method for Automatic Evaluation of Machine Translation*

[14] Matt Post *A Call for Clarity in Reporting BLEU Scores*

[15] Bhuwan Dhingra, Manaal Faruqui, Ankur Parikh, Ming-Wei Chang, Dipanjan Das, William W. Cohen *Handling Divergent Reference Texts when*

Evaluating Table-to-Text Generation