Task-free dialog system: a proposal

Felipe Salvatore

January 10, 2018

Abstract

to do

Contents

Chapter 1

Dialog Systems

1.1 intro

In this PhD thesis we will investigate the problem of **building a non-task-oriented dialog system**. This problem is central to the field of *artificial intelligence* (AI) due to the seminal paper by Alan Turing [18] in which he proposed an imitation game. The idea behind the game was simple: three players A, B and C can interact only by nonpersonal communication. C knows that he will interact with a computer (A) and a person (B), but he does not know which is which. C should exchange some conversation both with A and B; after some time he should guess who is the computer and who is the human. The goal of A is to be as human as possible in order to fool C; and the goal of B is to help C come to the right answer.

This is the famous *Turing Test*. At that time Turing proposed that test as a way to make more concrete the philosophical question "can machines think?". This can lead to a whole discussion of the nature of thinking and intelligence. As a mature field, AI has distantiate itself from these abstract an general questions and have concentrated in *building agents to solve tasks reserved exclusively to humans*: playing chess, driving a car, cleaning a room, writing a letter to a friend, etc.

The sub-area of AI that deals with problems involving human language is called *natural language processing* (NLP). It concentrate in tasks such text understanding (por mais coisa). NLP , as the field of AI as a whole, have change dramatically in the past 10 years. For some time the main paradigm in the field was the so called **knowledge base approach**: the main goal of AI was to develop intelligent systems capable of solving certain classes of problems by having a *representation* or *model* of the world. In this view, a decision by a machine is synonymous to *inferring in a formal language*[9].

A different point of view is that AI should construct systems that acquire their knowledge via data observation. More useful than programming hand-coded rules in an AI agent, we should enable that agent to extract patters from the complexity of data. This is the **machine learning approach**. One extension of this view is called **deep learning** where we learn i) how to map a representation of the data to a desirable output and ii) how to represent the data.

After this brief presentation we can state our objectives more clearly: here we will investigate how to build a non-task-oriented dialog system using a deep learning family of models. This family of models is know as **Recurrent Neural Networks** (RNNs). RNNs have achieve increasing success in a variety of NLP tasks such as machine translation, speech recognition, sentiment analysis, language modeling, etc. Together with this success the community has proposed also a variety of new architectures. We will explore some of these architectures for the task of dialog generation.

1.2 Theoretical framework

1.2.1 Machine learning

In particular for NLP it is usual to encode a linguistic feature as a dense vector. Linguistic features tent to be discrete entities (a word, a part-of-speech tag, etc.). So for some time the basic tool in the field was the use of 'one-hot vectors', i.e, a vector with only one entry being 1 and all the rest being 0. In this case each feature is its own dimension. The problem with that approach is that features become completely independent from one another. After the popularization of embeddings techniques [10] nowadays we let a model learn the best representation of a linguistic feature in some feature space in order to capture some correlation by the data.

1.2.2 Neural network

A neural network is a non-linear function $f(\boldsymbol{x};\theta)$. It is defined by a collection of parameters $\boldsymbol{\theta}$ and a collection of non-linear transformations. It is usual to represent f as a compositions of functions:

$$\begin{aligned} f(\boldsymbol{x};\theta) &= f^{(2)}(f^{(1)}(\boldsymbol{x};\boldsymbol{W}_1,\boldsymbol{b}_1);\boldsymbol{W}_2,\boldsymbol{b}_2) && (1.1)\\ &= softmax(\boldsymbol{W}_2(\sigma(\boldsymbol{W}_1\boldsymbol{x}+\boldsymbol{b}_1))+\boldsymbol{b}_2) && (1.2)\end{aligned}$$

The output of these intermediary functions are referred as *layers*. So in the example above, $\boldsymbol{x}$ (the output of the identity function) is the *input layer*, $f^{(1)}(\boldsymbol{x};\boldsymbol{W}_1,\boldsymbol{b}_1)$ is the *hidden layer* and $f^{(2)}(f^{(1)}(\boldsymbol{x};\boldsymbol{W}_1,\boldsymbol{b}_1);\boldsymbol{W}_2,\boldsymbol{b}_2)$ is the *output layer*. Since each layer is a vector, we normally speak about the *dimension* of a layer. For historical reasons we also say that each entry on a layer is a *node* or a *neuron*. Models with a large number of hidden layers are called *deep models*, for this reason the name *deep learning* is used.

A neural network is a function approximator: it can approximate any Borel measurable function from one finite dimensional space to another with any desired nonzero amount of error. This theoretical result is know as the *universal approximation theorem*[2]. Without entering in the theoretical concepts, it suffice to note that the family of Borel mensurable functions include all continuous functions on a closed and bounded subset of $\mathbb{R}^n$.

One thing to be noted is that the universal approximation theorem states that a neural network with at last one hidden layer with any 'squashing' activation function (such as the logistic sigmoid activation function) can be used to approximate the desired function. So it is natural to question why should anybody use a neural network with more than one hidden layer. The answer is that the theorem does not guarantee that a training algorithm will find the correct function generating on the data that we use for the training. Also a neural network with only one layer can use a hidden size of large dimension to represent one function, when a deep model can have more layers but with significantly smaller dimension. The main result in [17] makes this problem clear: for every positive integer k, there exist neural networks with $\Theta(k^3)$ layers, $\Theta(1)$ nodes per layer, and $\Theta(1)$ distinct parameters which can not be approximated by networks with $\mathcal{O}(k)$ layers and $o(2^k)$ nodes.

> Any time we choose a specific machine learning algorithm, we are implicitly stating some set of prior beliefs we have about what kind of function the algorithm should learn. Choosing a deep model encodes a very general belief that the function we want to learn should involve composition of several simpler functions.

> This can be interpreted from a representation learning point as saying that we believe the learning problem consist of discovering a set of underlying factors of variation that van in turn be described in terms of other, simpler underlying factors of variation. Alternately, we can interpret the use of a deep architecture as expressing a belief that the function we want to learn id a computer program consisting of multiple steps, where each step makes use of the previous step's output. These intermediate outputs are not necessarily factors of variation but can instead be analogous to counters or pointers that the network uses to organize its internal processing.[4, p. 195].

1.2.3 Computational graphs

1.3 Dialog as a learning task

important poins: i) Translation (MLE) vs MDP (policy gradients), ii) evaluation.

Dialogue is characterized by turn-taking: speaker A, then speaker B, etc.

We can view an utterance ina dialogue as a kind of action being performed by the speaker.

A program that generates dialog

to do

1.4 Dialog data

Twiter Ubuntu openSUb

1.5 Evaluation

about fooling people

BLUE, METEOR, others

Chapter 2

Generative models for text

Different deep learning architectures are used in NLP: **convolutional architectures** have a good performance in tasks were it is required to find a linguistic indicator regardless of its position (e.g., document classification, short-text categorization, sentiment classification, etc); high quality word embeddings can be achieved with models that are a kind of **feedforward neural network** [10]. But for a variety of works in natural language we want to capture regularities and similarities in a text structure. That is way **recurrent** and **recursive** models have been widely used in the field. Here we are focused on generative models and since recurrent models have been producing very strong results for language modeling [3], we will concentrate on them.

2.1 RNN

Recurrent Neural Network is a family of neural network specialized in sequential data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\tau$. As a neural network, a RNN is a parametrized function that we use to approximate one hidden function from the data. As before we can take the simplest RNN as a neural network with only one hidden layer. But now, what make RNNs unique is a recurrent definition of one of its hidden layer:

$$\boldsymbol{h}^{(t)} = g(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) \tag{2.1}$$

$\boldsymbol{h}^{(t)}$ is called *state*, *hidden state*, or **cell**.

Is costumerely to represent a RNN as a ciclic graph 2.1

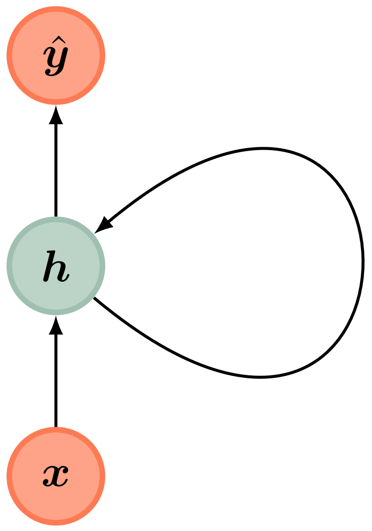

Figure 2.1: Ciclic representation

This recurrent equation can be unfolded for a finite number of steps τ. For example, when $\tau = 3$:

$$\begin{aligned} \boldsymbol{h}^{(3)} &= g(\boldsymbol{h}^{(2)}, \boldsymbol{x}^{(3)}; \boldsymbol{\theta}) && (2.2) \\ &= g(g(\boldsymbol{h}^{(1)}, \boldsymbol{x}^{(2)}; \boldsymbol{\theta}), \boldsymbol{x}^{(3)}; \boldsymbol{\theta}) && (2.3) \\ &= g(g(g(\boldsymbol{h}^{(0)}, \boldsymbol{x}^{(1)}; \boldsymbol{\theta}), \boldsymbol{x}^{(2)}; \boldsymbol{\theta}), \boldsymbol{x}^{(3)}; \boldsymbol{\theta}) && (2.4) \\ & && (2.5) \end{aligned}$$

Hence for any finite step τ we can describe the model as a DAG 2.1.

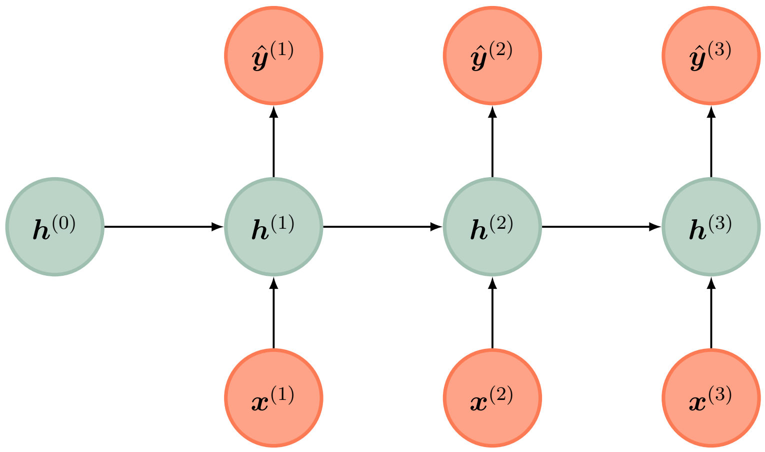

Figure 2.2: Unfolded computational graph

Using a concrete example consider the following model define by the equations:

$$f(\boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t-1)}; \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{U}, \boldsymbol{c}, \boldsymbol{b}) = \hat{\boldsymbol{y}}^{(t)} \tag{2.6}$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax(\boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}) \tag{2.7}$$

$$\boldsymbol{h}^{(t)} = g(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}) \tag{2.8}$$

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{b}) \tag{2.9}$$

Using the graphical representation the model can be view as:

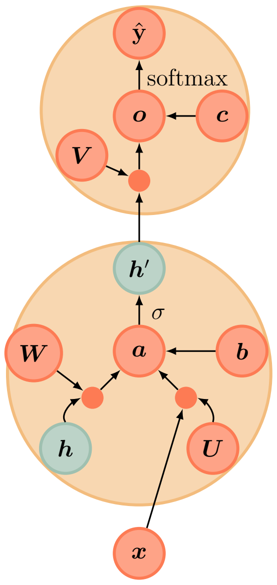

Figure 2.3: Graph of a RNN

!!! explicar como funciona o treinamento !!!

An RNN with a loss function:

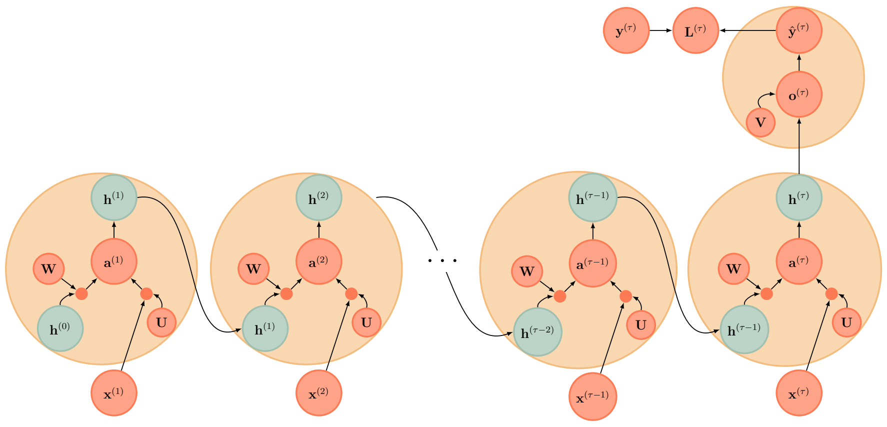

Figure 2.4: The computational graph to compute the training loss of a RNN

2.2 GRU

To capture long-term dependencies on a RNN the authors of the paper [1] proposed a new architecture called **gated recurrent unit** (**GRU**). This model was constructed to make each hidden state $\boldsymbol{h}^{(t)}$ to adaptively capture dependencies of different time steps. It work as follows, at each step t one candidate for hidden state is formed:

$$\widetilde{\boldsymbol{h}}^{(t)} = tahn(\boldsymbol{W}(\boldsymbol{h}^{(t-1)} \odot \boldsymbol{r}^{(t)}) + \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{b}) \tag{2.10}$$

where $\boldsymbol{r}^{(t)}$ is a vector with values in $[0, 1]$ called a **reset gate**, i.e., a vector that at each entry outputs the probability of reseting the corresponding entry in the previous hidden state $\boldsymbol{h}^{(t-1)}$. Together with $\boldsymbol{r}^{(t)}$ we define an **update gate**, $\boldsymbol{u}^{(t)}$. It is also a vector with values in $[0, 1]$. Intuitively we can say that this vector decides how much on each dimension we will use the candidate update. Both $\boldsymbol{r}^{(t)}$ and $\boldsymbol{u}^{(t)}$ are defined by $\boldsymbol{h}^{(t-1)}$ and $\boldsymbol{x}^{(t)}$; they also have specific parameters:

$$\boldsymbol{r}^{(t)} = \sigma(\boldsymbol{W}_r\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r\boldsymbol{x}^{(t)} + \boldsymbol{b}_r) \tag{2.11}$$

$$\boldsymbol{u}^{(t)} = \sigma(\boldsymbol{W}_u\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u\boldsymbol{x}^{(t)} + \boldsymbol{b}_u) \tag{2.12}$$

At the end the new hidden state $\boldsymbol{h}^{(t)}$ is defined by the recurrence:

$$\boldsymbol{h}^{(t)} = \boldsymbol{u}^{(t)} \odot \widetilde{\boldsymbol{h}}^{(t)} + (1 - \boldsymbol{u}^{(t)}) \odot \boldsymbol{h}^{(t-1)} \tag{2.13}$$

Note that the new hidden state combines the candidate hidden state $\widetilde{\boldsymbol{h}}^{(t)}$ with the past hidden state $\boldsymbol{h}^{(t-1)}$ using both $\boldsymbol{r}^{(t)}$ and $\boldsymbol{u}^{(t)}$ to adaptively copy and forget information.

It can appear more complex, but we can view the GRU model just as a refinement of the standard RNN with a new computation for the hidden state. Let $\boldsymbol{\theta} = [\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{b}]$, $\boldsymbol{\theta}_u = [\boldsymbol{W}_u, \boldsymbol{U}_u, \boldsymbol{b}_u]$ and $\boldsymbol{\theta}_r = [\boldsymbol{W}_r, \boldsymbol{U}_r, \boldsymbol{b}_r]$; and $\text{aff}(\boldsymbol{\theta})$ be the following operation:

$$\text{aff}(\boldsymbol{\theta}) = \boldsymbol{W}\boldsymbol{h} + \boldsymbol{U}\boldsymbol{x} + \boldsymbol{b} \tag{2.14}$$

With similar definitions for $\text{aff}(\boldsymbol{\theta}_u)$ and $\text{aff}(\boldsymbol{\theta}_r)$. Figure 2.2 shows the hidden state of the GRU model for time step t. If compared with Figure 2.1 we can see that the basic structure is the same, just the way of computing the hidden state $\boldsymbol{h}^{(t)}$ has changed.

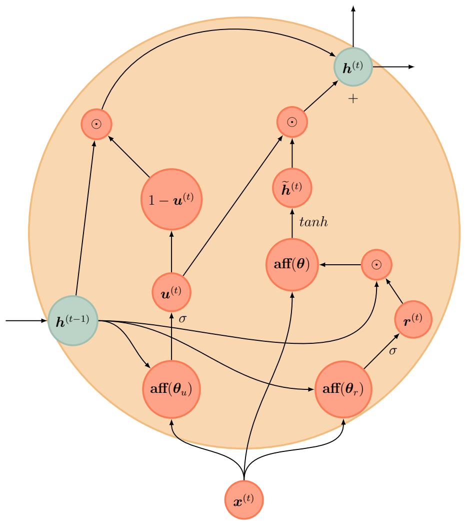

Figure 2.5: GRU hidden cell

2.3 LSTM

2.4 Language model

We call *language model* a probability distribution over sequences of tokens in a natural language.

$$P(x_1, x_2, x_3, x_4) = p$$

This model is used for different nlp tasks such as speech recognition, machine translation, text auto-completion, spell correction, question answering, summarization and many others.

The classical approuch to a languange model was to use the chain rule and a markovian assumptiom, i.e., for a specific n we assume that:

$$P(x_1, \dots, x_T) = \prod_{t=1}^{T} P(x_t | x_1, \dots, x_{t-1}) = \prod_{t=1}^{T} P(x_t | x_{t-(n+1)}, \dots, x_{t-1}) \quad (2.15)$$

This gave raise to models based on n-gram statistics. The choice of n yields different models; for example *Unigram* language model ($n = 1$):

$$P_{uni}(x_1, x_2, x_3, x_4) = P(x_1)P(x_2)P(x_3)P(x_4) \tag{2.16}$$

where $P(x_i) = count(x_i)$.

Bigram language model ($n = 2$):

$$P_{bi}(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_2)P(x_4|x_3) \tag{2.17}$$

where

$$P(x_i|x_j) = \frac{count(x_i, x_j)}{count(x_j)}$$

Higher n-grams yields better performance. But at the same time higher n-grams requires a lot of memory[5].

Since [11] the approach has change, instead of using one approach that is specific for the language domain, we can use a general model for sequential data prediction: a RNN.

So, our learning task is to estimate the probability distribution

$$P(x_n = \text{word}_{j^*}|x_1, \ldots, x_{n-1})$$

for any $(n-1)$-sequence of words $x_1, \ldots, x_{n-1}$.

We start with a corpus C with T tokens and a vocabulary $\mathbb{V}$.

Example: **Make Some Noise** by the Beastie Boys.

Yes, here we go again, give you more, nothing lesser
Back on the mic is the anti-depressor
Ad-Rock, the pressure, yes, we need this
The best is yet to come, and yes, believe this
...

- $T = 378$
- $|\mathbb{V}| = 186$

The dataset is a collection of pairs $(\boldsymbol{x}, \boldsymbol{y})$ where $\boldsymbol{x}$ is one word and $\boldsymbol{y}$ is the immediately next word. For example:

$(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}) =$ (Yes, here).

$(\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}) =$ (here, we)

$(\boldsymbol{x}^{(3)}, \boldsymbol{y}^{(3)}) =$ (we, go)

$(\boldsymbol{x}^{(4)}, \boldsymbol{y}^{(4)}) =$ (go, again)

$(\boldsymbol{x}^{(5)}, \boldsymbol{y}^{(5)}) =$ (again, give)

$(\boldsymbol{x}^{(6)}, \boldsymbol{y}^{(6)}) =$ (give, you)

$(\boldsymbol{x}^{(7)}, \boldsymbol{y}^{(7)}) =$ (you, more)

...

Notation

- $\boldsymbol{E} \in \mathbb{R}^{d,|\mathbb{V}|}$ is the matrix of word embeddings.
- $\boldsymbol{x}^{(t)} \in \mathbb{R}^{|\mathbb{V}|}$ is one-hot word vector at time step t.
- $\boldsymbol{y}^{(t)} \in \mathbb{R}^{|\mathbb{V}|}$ is the ground truth at time step t (also an one-hot word vector).

The language model is similar as the RNN described above. It is defined by the following equations:

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)} \tag{2.18}$$

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{e}^{(t)} + \boldsymbol{b}) \tag{2.19}$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax(\boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}) \tag{2.20}$$

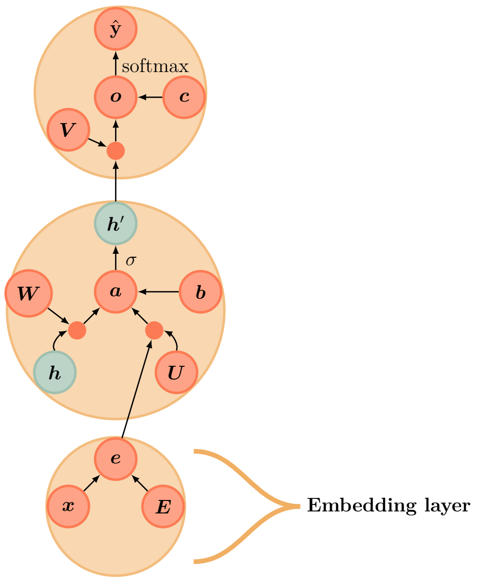

Figure 2.6: Simple language model

At each time t the point-wise loss is:

$$
\begin{aligned}
L^{(t)} &= CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) && (2.21)\\
&= -\log(\hat{\boldsymbol{y}}_{j^*}) && (2.22)\\
&= -\log P(x^{(t+1)} = \text{word}_{j^*} | x^{(1)}, \ldots, x^{(t)}) && (2.23)
\end{aligned}
$$

The loss L is the mean of all the point-wise losses

$$L = \frac{1}{T} \sum_{t=1}^{T} L^{(t)} \tag{2.24}$$

Evaluating a language model. We can evaluate a language model using a *extrinsic evaluation*: How our model perform in a NLP task such as text auto-completion. Or a *intrinsic evaluation*: Perplexity (PP) can be thought as the weighted average branching factor of a language.

Given $C = x_1, x_2, \ldots, x_T$, we define the perplexity of C as:

$$PP(C) = P(x_1, x_2, \ldots, x_T)^{-\frac{1}{T}} \tag{2.25}$$
$$\tag{2.26}$$
$$= \sqrt[T]{\frac{1}{P(x_1, x_2, \ldots, x_T)}} \tag{2.27}$$
$$\tag{2.28}$$
$$= \sqrt[T]{\prod_{i=1}^{T} \frac{1}{P(x_i|x_1, \ldots, x_{i-1})}} \tag{2.29}$$

we can relate Loss and Perplexity:
Since

$$L^{(t)} = -\log P(x^{(t+1)}|x^{(1)}, \ldots, x^{(t)}) \tag{2.30}$$
$$= \log(\frac{1}{P(x^{(t+1)}|x^{(1)}, \ldots, x^{(t)})}) \tag{2.31}$$
$$\tag{2.32}$$

We have that:

$$L = \frac{1}{T}\sum_{t=1}^{T} L^{(t)} \tag{2.33}$$
$$= \log\left(\sqrt[T]{\prod_{i=1}^{T} \frac{1}{P(x_i|x_1, \ldots, x_{i-1})}}\right) \tag{2.34}$$
$$= \log(PP(C)) \tag{2.35}$$

So another definition of perplexity is

$$2^L = PP(C) \tag{2.36}$$

2.5 Encoder Decoder

2.6 Atention

Chapter 3

Archictetures for dialog

3.1 Build End-to-End Dialogue Systems

The strategy in [15] is to train a dialogue system using only a question-answer corpus, i. e., they will train a generative model to generate one utterance when another one is given (and a context is given too). The main supposition is that the response generation problem can be seen as a translation problem. Generate a sentence in a dialogue is just translate the question into a response.

In this setting a dialogue is viewed as a sequence of utterances $D = \langle U_1, \dots, U_M \rangle$ involving two interlocutors such that each U_i contains a sequence of N_i tokens, $U_i = \langle w_{i,1}, \dots, w_{i,N_i} \rangle$. Each $w_{i,j}$ is a random variable that taking values in the $V \cup A$ where V is the vocabulary and A is the set of *speech acts* (for example 'pause' and 'end of turn' are speech acts). In this case the learning task is the same as the one in language modeling: we want to estimate of the probability of one token (speech acts included) conditioned on the previously tokens:

$$P(w_{i,j} | w_{i,1}, \dots, w_{i,j-1}, U_1, \dots U_{i-1}) \tag{3.1}$$

The different dialogues $\langle U_1, \dots, U_M \rangle$ will serve as a corpus. In this paper the authors use a **Hierarchical Recurrent Encoder-Decoder model(HRED)**, i.e., a model based on three RNNs: an *encoder* RNN, a *context* RNN and a *decoder* RNN. To understand the workings of this model, suppose we have the following dialog:

- (U_1) Mom, I don't feel so good.
- (U_2) What's wrong?
- (U_3) I feel like I'm going to pass out.

First the encoder RNN will encode the sentence U_1 in a vector $\boldsymbol{U}_1$, then the context RNN will compute a hidden state, say $\boldsymbol{C_1}$ using $\boldsymbol{U}_1$ and all past sentences in the dialog. After that the decoder RNN will use U_2 as an input to predict the next word using $\boldsymbol{C_1}$. The same procedure is repeated for U_2. Figure x shows the computational graph of this process.

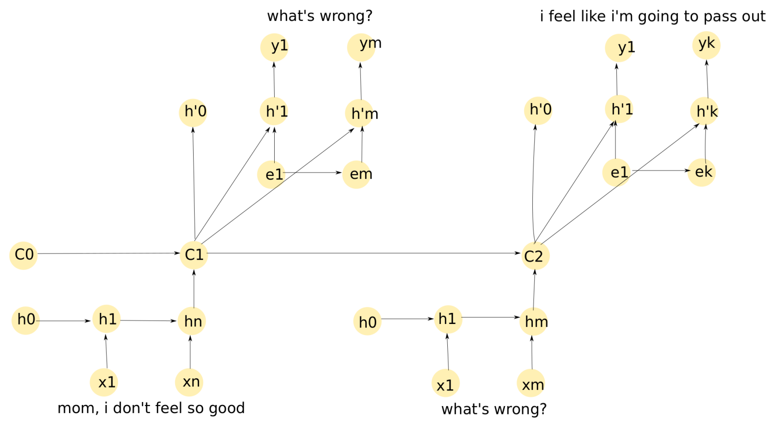

Figure 3.1: HRED model

Note that in this model the prediction is conditioned on the hidden state of the context RNN.

In this paper the authors use the world perpelexity from a part of the corpus to evaluate the language model.

The learning is based on maximized the log likelihood of the language model (the parametrized probabilistic distribution), one of the results is that model learn to predict the most frequent utterances like 'I don't know' or 'I'm sorry'.

3.2 Multiresolution Recurrent Neural Networks

In the paper [13]

3.3 3

3.4 4

3.5 Deep Reinforcent Learning

In the paper *Deep Reinforcement Learning for Dialogue Generation* [6] the authors notice that since seq2seq models are trained by predicting the next sentence in a given conversation using the maximum-likelihood estimation (MLE) objective function, the model tends to generate highly generic responses such as "*I don't know*". This kind of responses creates conversation with little or no informational content. For example, the author let two agents talk and find some unsatisfactory results:

A) how old are you?

B) I'm 16

A) 16?

B) I don't know what are you talking about.

A) You don't know what are you saying.

B) I don't know what are you talking about.

A) You don't know what are you saying.

...

With that in mind the authors propose adding a rewarding signal to to guide the agent towards more meaningful responses, and formulate the dialog problem such that it is possible to model the long term influenced of a utterance. They choose the theoretical framework of reinforcement learning to achieve that.

This is a way to change the objective function that is being maximized during training. Instead of MLE objective, they try to learn a policy optimizing the long-term developer-defined reward from ongoing dialog simulations using policy gradient methods.

The authors are trying to combine two traditions of research in dialog generation, the one that sees dialog generation as a translation problem (the program should translate one utterance in a response); and the other that are focused in task oriented dialogs, an agent that utter a sentence is view as an agent that need to take action in an environment. This second tradition models the dialog task as a reinforcement learning problem.

In this paper the learning system is described as follows: there are to agents A and B. They take turn talking with each other. So a dialogue can be represented as the sequence: $p_1, q_1, p_2, q_2, \dots, p_i, q_i$ where p and q are the sentences uttered by A and B, respectively.

For any agent, and **action** a is a sentence. Since sentence can have any length, the action space is infinite.

A **state** is a pair (p_i, q_i) of the previous two dialogue turns.

A stochastic policy $\pi(a|s)$ takes the form of an LSTM encoder-decoder $P(p_{i+1}|p_i, q_i)$.

The reward system in this setup have many parts. i) **Ease of answering** the first part of the reward is the signal for *forward-looking*, i.e., the agent should be punished if he produce sentences that do not move the the conversation forward. Using a set of S of sentences such as 'I don't know what you are talking about', 'I have no idea', etc. and a Seq2Seq model $p_{Seq2Seq}$, a reward signal is calculated as

$$r_1 = -\frac{1}{N_S} \sum_{s \in S} \frac{1}{N_s} \log p_{Seq2Seq} \tag{3.2}$$

where a is an action (a sentence) N_S is the cardinality of S and N_s is the number of tokens in the dull sentence s.

ii) **Information Flow** the authors penalize semantic similarity (cosine similarity) between consecutive turns form the same agent. Let p_i and p_{i+1} be two consecutive turns and h_{p_i} and $h_{p_{i+1}}$ be the respective representation obtained form the encoder, them the reward signal r_2 is defined as the negative log of the cosine similarity:

$$r_2 = -\log \frac{h_{p_i} h_{p_{i+1}}}{||h_{p_i}||||h_{p_{i+1}}||} \tag{3.3}$$

iii) **Semantic Coherence** the authors also consider the mutual information between the action a and previous turns in the history to ensure the generated responses are coherent and appropriate:

$$r_3 = \frac{1}{N_a} \log p_{Seq2Seq}(a|q_i, p_i) + \frac{1}{N_{q_i}} \log p^{backward}_{Seq2Seq}(q_i|a) \tag{3.4}$$

where $p_{Seq2Seq}(a|p_i, q_i)$ denotes the probability of generating response a given the previous dialogue utterances $[p_i, q_i]$. $p^{backward}_{Seq2Seq}(q_i|a)$ denotes the backward probability of generating the previous dialogue utterance q_i based on response a.

And so the reward for action a is:

$$r(a, [p_i, q_i]) = \lambda_1 r_1 + \lambda_2 r_2 + \lambda_3 r_3 \tag{3.5}$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$ (the authors have used $\lambda_1 = 0.25, \lambda_2 = 0.25, \lambda_3 = 0.5$.

First the autors have trained the Seq2Sqeq model on the OpenSubtitles dataset; the treat each turn in the dataset as a target and the concatenation of two previous sentences as source inputs. they use this trained model to train a mutial information model. This last model is the one that they use to initialize the policy model p_{RL}.

In this context an episode is a simulate conversation between two agents (A and B). At the initial step a message from the training set is fed to A. Using an encoder -decoder model, A encodes the message and decode a set of responses to B. In its turn, B combines the set of immediate output of A with the dialog history and using the encoder-decoder model generates a set of responses. These responses are feeded to A, and the process continues. Figure 3.5 shows one visualization of this process. Each played action is considered a *turn*. There are two stopping criteria: one of the agents generate dull responses like "I don't know" or two consecutive responses from the same agent are highly overlapping (i.e., they share more than 80% of their words).

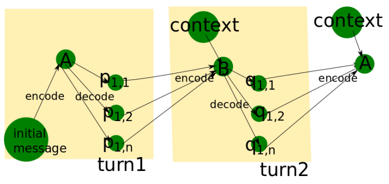

Figure 3.2: MDP conversation

The authors use policy gradient methods to search a model to maximize the expected future reward:

$$J_{RL}(\boldsymbol{\theta}) = \mathbb{E}_{p_{RL}(a_{1:T})}\left[\sum_{i=1}^{T} R(a_i, [p_i, q_i])\right] \tag{3.6}$$

where $R(a_i, [p_i, q_i])$ denotes the reward resulting from action a_i. The gradient is estimated by the *likelihood ratio trick*:

$$\nabla_{\boldsymbol{\theta}} J_{RL} \approx \sum_i \nabla_{\boldsymbol{\theta}} p(a_i|p_i, q_i) \sum_{i=1}^{T} R(a_i, [p_i, q_i]) \tag{3.7}$$

Three different metrics are used: i) *length of the dialog* – average number of simulated turns; ii) *degree of diversity* – number of distinct unigrams and bigrams in generated responses; iii) *human evaluation.*

The paper compare three models: Seq2seq, mutual information and the proposed model using reinforcement learning (RL model). The RL model have a slightly better score then the mutual information model in i) and ii). But in iii) human judges are presented with simulated conversation between the two agents, some conversation were generated by the RL model and some were generated by the mutual information model. The judges voted in which one is the better (ties are permitted). The RL model win 72% of time, lose 12% of time and tie 16% of time.

3.6 How to evaluate dialogs?

wewewe

BLUE

This metric was proposed in [12] for automatic translation. It compares n-grams (up to 4) of the candidate translation with the n-grams of the reference translation and count the number of matches; it also ads brevity penalty for too short translations. The formula for this metrics is:

$$BLUE = min\left(1, \frac{output - length}{reference - lenght}\right)\left(\prod_{n=1}^{4} precision_n\right)^{\frac{1}{4}} \tag{3.8}$$

Where $precision_n$ is number of n-gram overlap between the candidate and the reference divided by the number of all n-grams in the candidate. BLUE scores ranges from 0 to 1. Typically this score is computed over an entire corpus and was originally designed for use with multiple reference sentences. To give one simple example we will use the following Portuguese sentence:

Metric	c_1	c_2
$precision_1$	5/7	4/7
$precision_2$	3/6	2/6
$precision_3$	2/5	1/5
$precision_4$	1/4	0/4
brevity penalty	7/8	7/8
$BLUE$	0.38	0

'em plano aberto, a cidade parece linda'

The reference translation is

'in a wide shot, the city looks beautiful'

Now consider two candidates:

c_1: 'in the open, the city looks beautiful'
c_2: 'in open plan, the city looks gorgeous'

Table 3.6 shows the precision for each n-gram (up to 4) and the BLUE score for each candidate.

Automatic Turing test

The authors of [8] defend that the Turing Test [18] provides one accurate automatic evaluation procedure for dialog systems. But the test need to be changed in order to avoid human evaluation. Theirs basic assumption is that : *a good chatbot is one whose responses are score highly on appropriateness by human evaluators*. So they collected a dataset of appropriateness scores to various dialogues responses; with this dataset they have trained a model (called automatic dialogue evaluation model - ADEM) to predict human scores.

The dataset contains 4104 conversational responses confectioned on dialog contexts and human judgments (scores). The context is from the Twitter Corpus and the responses are generated by different models (these are also human-generated responses but they are not the ones from the original corpus).

BLEU is a metric to compute word overlap. It is normally used for automatic translation. It can be used to dialogue response evaluation but it fail in capturing the semantic similarity between the model and reference responses. And since this method compares only the model's output and the reference response, it does not consider the context of the conversation.

ADEM is a hierarchical RNN encoder. Given the dialogue context c, reference response r and model response $\hat{r}$ ADEM encodes each one of them into a vector ($\boldsymbol{c}, \boldsymbol{r}$ and $\hat{\boldsymbol{r}}$ respectively) and computes the following score:

$$score(c, r, \hat{r}) = \frac{(\boldsymbol{c}^T \boldsymbol{M} \hat{\boldsymbol{r}} + \boldsymbol{r}^T \boldsymbol{N} \hat{\boldsymbol{r}} - \alpha)}{\beta} \tag{3.9}$$

The matrices $\boldsymbol{M}$ and $\boldsymbol{N}$ map the model response $\hat{\boldsymbol{r}}$ into the space of contexts and reference responses, respectively. The model gives high scores to responses that have similar vector representations to the context and reference response after this projection. The model is trained in order to minimize the squared error between the model prediction and the human score with L2-regularization:

$$J(\boldsymbol{M}, \boldsymbol{N}) = \sum_{i}^{K} [score(c, r, \hat{r}) - human_i]^2 + \gamma(||\boldsymbol{M}||^2 + ||\boldsymbol{N}||^2) \tag{3.10}$$

To have a good encoder, the authors pre-trained the model (only the parameters of the encoder) using a dialogue model, more specifically a latent variable hierarchical recurrent encoder-decoder (VHRED) model.

Using this model the authors obtained a good correlation between ADEM and the human judgment. The model is often very good at assigning low scores to poor responses, but it tends to be too conservative because the model learns to predict scores closer to the average human score.

The authors notice that the metrics from [6] are based on hand-crafted features and it is unclear whether such objectives are preferable over retrieval-based cross-entropy or word-level log-likelihood objectives. They also notice that current dialogue systems are incapable of generating responses that are rated as highly appropriate by humans.

Bibliography

[1] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[2] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2:303–314, 1989.

[3] Y. Goldberg. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015.

[4] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2017.

[5] K. Heafield. Scalable modified kneser-ney language model estimation. In *In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013.

[6] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky. Deep reinforcement learning for dialogue generation. *CoRR*, abs/1606.01541, 2016.

[7] J. Li, W. Monroe, T. Shi, A. Ritter, and D. Jurafsky. Adversarial learning for neural dialogue generation. *CoRR*, abs/1701.06547, 2017.

[8] R. Lowe, M. Noseworthy, I. V. Serban, N. Angelard-Gontier, Y. Bengio, and J. Pineau. Towards an automatic turing test: Learning to evaluate dialogue responses. *CoRR*, abs/1708.07149, 2017.

[9] J. McCarthy and P. J. Hayes. Readings in nonmonotonic reasoning. chapter Some Philosophical Problems from the Standpoint of Artificial Intelligence, pages 26–45. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[11] T. Mikolov, S. Kombrink, L. Burget, J. Cernocký, and S. Khudanpur. Extensions of recurrent neural network language. *IEEE*, pages 5528–5531, 2011.

[12] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: A method for automatic evaluation of machine translation. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics, 2001.

[13] I. V. Serban, T. Klinger, G. Tesauro, K. Talamadupula, B. Zhou, Y. Bengio, and A. C. Courville. Multiresolution recurrent neural networks: An application to dialogue response generation. *CoRR*, abs/1606.00776, 2016.

[14] I. V. Serban, R. Lowe, L. Charlin, and J. Pineau. Generative deep neural networks for dialogue: a short review. *CoRR*, abs/1611.06216, 2016.

[15] I. V. Serban, A. Sordoni, Y. Bengio, A. Courville, and J. Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the Thirtieth AAAI Conference of Artificial Intelligence*, AAAI'16, pages 3776–3783. AAAI Press, 2016.

[16] I. Sustskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, 2014.

[17] M. Telgarsky. Benefits of depth in neural networks. *CoRR*, abs/1602.04485, 2016.

[18] A. Turing. Computing machinery and inteligence. *Mind*, pages 433–460, 1950.

[19] T. Wen, M. Gasic, N. Mrksic, L. M. Rojas-Barahona, P. Su, S. Ultes, D. Vandyke, and S. J. Young. A network-based end-to-end trainable task-oriented dialogue system. *CoRR*, abs/1604.04562, 2016.

[20] Z. Yu, Z. Xu, A. W. Black, and A. I. Rudnicky. Strategy and policy learning for non-task-oriented conversational systems. In *Proceedings of the SIGDIAL 2016 Conference, The 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue, 13-15 September 2016, Los Angeles, CA, USA*, pages 404–412. The Association for Computer Linguistics, 2016.