# Project 2 - E2E Dialogue Model

### Goal:

Design and train an end-to-end, task-oriented,multi-turn, generative dialogue model that provides technical support for the Ubuntu operating system.

### Index:

1. Introduction

2. Litterature Review

3. Files and Folders - installation and description of the files and the folders in the project

4. Dataset and Data Preparation

5. Baseline model - LSTM-based

5.1 GRU + GRU + Attention

5.2 biLSTM + GRU + Attention

6. Transformers-based model

6.1 Self Attention

6.2 Hierarchical Self Attention

6.3 Beam Search vs Contrastve Search

7. GPT2 Language Model

8. Evaluation of Dialogue Models

9. Inference

10. Results and Discussion

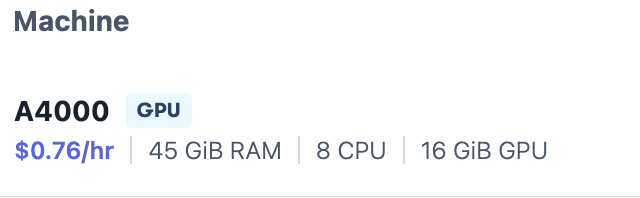
11. Conclusion and Future Directions

12. References

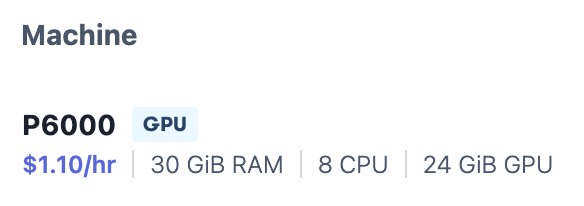
13. Acknowledgement

# 1. Files and folders

* checkpoints\_big: folder with the checkpoints for the trained model with the recommended setup from the paper 'Attention is all you need' and the big dataset
* checkpoints\_small: folder with the checkpoints for the trained model with the small architecture and the small dataset
* json\_big: folder containing one file with the big dataset already transformed to json by the PolyAI script
* json\_small: folder containing a set of files with the small dataset. This is splitted into different files as this is how the PolyAI script works by default. The test files are not used
* clean\_data.py: script executing a bash command for the create\_data.py and its options. I had to do it this way in order to avoid having to modify all the script
* create\_data.py: modified version of the PolyAI script to create the JSON files from the Opensubtitles dataset. I did a small modification to solve one problem when encoding one of the files name and also modifying some of the default parameters values as it didn't seem to work when calling them from the command line
* evaluate.py: not a trully 'evaluation'. What it does is give the response to a given sentence. It first creates the dictionary of tokens from the dataset, loads the checkpoint and generates the response. See below how to change from the small dataset to the big one
* model.py: the model parts implementation from [tensorflow](https://www.tensorflow.org/tutorials/text/transformer)
* test.json: sample file of the output from the PolyAI script
* tokenizer.py: script to generate the dictionary from the json records used for training and evaluation
* train.py: the actual model training



For bidirectional LSTM



# 2. Introduction

The advancements in deep neural networks allow us to build end-to-end dialogue models, which is a single system of jointly trained networks that can accomplish all the tasks necessary to generate a dialogue.

When we speak of an ‘end-to-end’ dialogue system, we mean a single system that can be used

to solve each of these four aspects simultaneously (see Figure 1). Typically this is a system that

takes as input the history of the conversation and is trained to optimize a single objective, which

is a function of the textual output produced by the system and the correct (ground truth) response.

This is in contrast to the ‘modular’ system approach to dialogue systems, where each component of Figure 1 is trained separately, and either takes a more structured input, such as a set of dialogue acts, or is trained to maximize an intermediary objective, such as slot-filling.

a modular dialogue system as a system where two or more elements (sub-components or system parameters) are optimized with respect to two or more different objective functions (e.g. where the State Tracker is trained to minimize the cross-entropy error of predicting the slot-value pairs, and where the Response Generator is trained to maximize the conditional log-likelihood of the correct response given the slot-value pairs). Thus, any machine learning-based dialogue system which is not a modular dialogue system is an end-to-end dialogue system.

In previous labs, we have implemented machine translation, which is used to read the source language (input) and generate the desired language (output). Similarly, in a dialogue system, we will implement a model to generate a response given a context. This is also known as Natural Language Generation (NLG).

<http://cs230.stanford.edu/projects_winter_2019/reports/15811573.pdf>

Challenges and how we addressed them

Long sequences => memory requirements, vanishing gradient

Specific pre-processig for dialogue

Specific evaluation for dialogie

Deal with large datasets

Training DNN models is often very time-consuming and expensive

Measured applied

Used gradient clipping

Reduced learning rate - learning rate annealing

Reduced batch size

Reduced vocabulary size

Sort the batch

training of the model on a **large dataset**.

We train the model with tons of data

But we simplify the network

It is self-supervised problem, you don’t need human annotated data

Note: ASR needs annotated data

Scaling up means more parameters , larger datasets

Note: the models are such that they allow scaling up

You need a model that scales up efficiently

Transformer is such a model

It is able to digest a lot of data

Parallelization requires memory => it is not a small footprint model

You can do distributed training on many GPU's

At training time all the encoding steps are in parallel

Transformers can learn long term dependencies through self-attention and cross attention

Also add shortcuts

Note: GRU can also learn long term dependencies

NOTE: LM benefits a lot from understanding long-term dependencies.

PLAN / APPROACH

End-2-end

Encoder-decoder

An encoder is a type of neural network architecture that is used to transform raw input data into a **useful representation** . use an encoder-decoder architecture to generate natural language text, where the encoder is responsible for encoding the input sequence and the decoder generates the output text.

RNN can learn long history

Explore RNN and Transformer architectures for generative dialogue

There is no one "best" neural network architecture for dialogue, as the optimal architecture may vary depending on the specific task, dataset, and other factors. However, here are some commonly used architectures for dialogue systems:

1. Sequence-to-Sequence (Seq2Seq) Models: Seq2Seq models use an encoder-decoder architecture to generate a response given an input utterance. These models have been successfully used for chatbot and conversational agent applications.
2. Transformer Models: Transformer models, such as the ones used in GPT and BERT, have been shown to perform well on language modeling and dialogue generation tasks. These models use a self-attention mechanism to process the input sequence and generate a response.
3. Hierarchical Models: Hierarchical models use a multi-level structure to capture the context and structure of dialogue. These models have been used to generate longer and more coherent responses in conversational systems.
4. Memory Networks: Memory networks use an external memory component to store information and retrieve relevant information during dialogue. These models have been used for tasks such as question answering and dialogue management.

It is important to note that the effectiveness of these models may depend on factors such as the size and quality of the training data, the complexity of the dialogue task, and the evaluation metrics used. Therefore, it is important to carefully consider the specific requirements and constraints of the dialogue task before selecting a neural network architecture.

Designing a generative dialogue model involves several steps, including data collection, pre-processing, model architecture selection, training, and evaluation. Here is a general outline of the steps involved in designing a generative dialogue model:

1. Data Collection: Collect a large dataset of dialogue examples that are relevant to the task. This dataset can be obtained from various sources, such as social media, chat logs, or specialized datasets for specific dialogue tasks.
2. Data Pre-processing: Pre-process the dataset by cleaning the data, tokenizing the text, and converting it into a numerical format that can be used as input to the model. It is also important to split the data into training, validation, and test sets.
3. Model Architecture Selection: Select a suitable model architecture for the task. Commonly used architectures for generative dialogue include sequence-to-sequence (Seq2Seq) models, Transformer models, hierarchical models, and memory networks. The choice of architecture will depend on the specific requirements and constraints of the task.
4. Training: Train the selected model architecture on the pre-processed dataset using a suitable optimization algorithm, such as stochastic gradient descent (SGD). During training, it is important to monitor the loss function and adjust the hyperparameters, such as learning rate, batch size, and number of epochs, to optimize performance.
5. Evaluation: Evaluate the performance of the trained model on the validation and test sets using appropriate metrics, such as perplexity, BLEU score, or human evaluation. It is important to ensure that the model is generating high-quality responses that are relevant to the task and that demonstrate coherence and fluency.
6. Deployment: Deploy the trained model in a production environment, such as a chatbot or conversational agent, and continue to monitor performance and make improvements as necessary.

Overall, designing a generative dialogue model requires careful consideration of the data, task, and model architecture, as well as attention to detail in pre-processing, training, and evaluation.

Task-oriented generative dialogue systems require a neural network architecture that can generate responses that are not only fluent and coherent but also informative and goal-oriented. Here are some neural network architectures that are suitable for task-oriented generative dialogue:

1. Sequence-to-Sequence (Seq2Seq) Models: Seq2Seq models have been successfully used for task-oriented dialogue systems, where the model generates responses that are relevant to a specific task or domain. These models can be trained on a dataset of dialogue examples that include input utterances and corresponding output actions or responses.
2. Encoder-Decoder Models with Attention: Attention mechanisms can be added to Seq2Seq models to improve the model's ability to focus on relevant information and generate informative responses. These models can be trained to map input utterances to output actions or responses that are specific to a given task or domain.
3. Transformer Models: Transformer models, such as BERT and GPT, have been shown to perform well on natural language processing tasks, including task-oriented dialogue systems. These models can be fine-tuned on a task-specific dataset to generate informative and coherent responses that are relevant to the task.
4. Hierarchical Models: Hierarchical models use a multi-level structure to capture the context and structure of dialogue. These models can be trained to generate informative and goal-oriented responses by incorporating task-specific information at different levels of the hierarchy.
5. Memory Networks: Memory networks use an external memory component to store and retrieve relevant information during dialogue. These models can be trained to generate informative and task-oriented responses by incorporating task-specific information into the memory component.

In summary, the choice of neural network architecture for task-oriented generative dialogue depends on the specific requirements and constraints of the task, as well as the available data and resources. However, Seq2Seq models with attention, Transformer models, hierarchical models, and memory networks are all suitable architectures for task-oriented generative dialogue.

Future Steps:

Scaling up

The Neural Network learns this kind of relationships automatically WITH A LOT OF DATA

* They scale up the model both in terms of number of **parameters** (capacity) and amount of training **data**.

# 3. Litterature Review

Generating models: • Hierarchical recurrent encoder-decoder (HRED)

traditional dialog systems are divided into retrieval-based and generative systems. The former needs to select appropriate responses from a set of candidate facts to match user requests [[3](https://www.frontiersin.org/articles/10.3389/fphy.2022.1019969/full#B3)], while the latter directly generates more free responses according to the questions.

HRED. Finally we consider the Hierarchical Recurrent Encoder-Decoder (HRED) (Serban et al., 2015). **In the traditional Encoder-Decoder framework, all utterances in the context are concatenated together before encodin**g. Thus, information from previous utterances is far outweighed by the most recent utterance. The HRED model uses a hierarchy of encoders; each utterance in the context passes through an ‘utterance-level’ encoder, and the

Dialog- BERT employs a hierarchical Transformer architecture to represent the dialogue context. It first encodes dialogue utter- ances through a Transformer encoder and then encodes the re- sulting utterance vectors using a discourse-level Transformer to obtain a representation of the entire dialogue context.

# 4. Dataset

The Ubuntu Dialogue Corpus [2] consists of one million two-person conversations extracted from the Ubuntu technical support forum which makes it suitable for task-oriented training. The style of the messages is closely correlated to natural human-to-human dialogue, however, there are a lot of special symbols and technical terms. The conversations have on average 8 turns each, with a minimum of 3 turns which helps the modelling of longer-term dependencies. The average dialogue history is 86 words long and the average utterance is 17 words long. Additional pre-processing tasks like named entity resolution have been performed to improve the model's performance.

In the raw dataset, each line represents a turn in the conversation between speaker1 and speaker2 [Figure1]. We consider that the conversation is always initiated by the human user (speaker1) and the reply is generated by the system (speaker2)

2005-02-28" 
2005-02-28T13:07 :OO.OOOZ 
2005-02-28" 
:oo.oooz 
2005-02-28" 
2005-02-28T13:57 :OO.OOOZ 
2005-03-02T09:41 :OO.OOOZ 
ells 
tritium 
ells 
tritium 
tritium 
ells 
tritium 
tritium 
ells 
tritium 
ells 
ells 
tritium 
ells 
ells 
tritium 
ells 
ells 
tritium 
tritium 
ells 
tritium 
ells 
ells 
tritium 
ells 
ells 
tritium 
ells 
tritium 
tritium 
ells 
tritium 
tritium 
ells 
tritium 
tritium 
ells 
what is up 
hello Just getting ready to sleep. You? 
got a quick question 
rm -rf dir/ 
use chmod to change the permissions 
my keyboard for my laptop intermittently doing crazy stuff. Is there a way i 
you could try • reset' 
wouldn't you rather use the gnome keyboard configuration tool? 
I got a question for you 
0k 
When I hooked up my hard drive to the ribbon, it would not boot, when I unhc 
or if you have an idea 
did you hook the drive up correctly? 
honestly, I think so 
could there be something wrong with the board. It just is crazy that the 
is the drive new and possibly unsupported by the moboQ 
the computer wont boot with either the cdrom or hard drive hooked up. 
I am leaning to the mother board somehow being bad 
sounds possible. 

Figure1: Raw dataset

We have to represent the raw input text in a structured format that can be processed by the model. The authors chose to concatenate all consecutive utterances by the same user into one utterance and put \_\_EOS\_\_ at the end (Figure2).

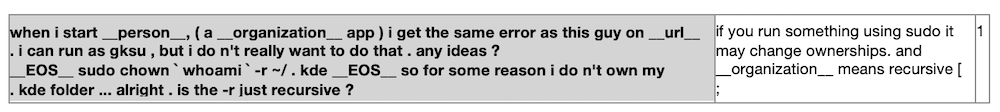


Figure2. Preprocesed dataset

The disadvantage of this representation is that we lose track of the speaker. In the second column we want to have the reply from the system (speaker2) so that we can train the model to play the role of speaker2.

That is why we slightly changed the format of the input.

We could have the speaker name coupled with the corresponding utterance in a key-value pair.

Representing the input for a dialogue model involves converting the raw input text into a structured format that can be processed by the model. Here are some common techniques for representing the input:

1. Tokenization: The input text is split into individual words or sub-words, also known as tokens.
2. Embedding: Each token is converted into a vector representation, known as an embedding. This allows the model to represent the meaning of each token in a high-dimensional space.
3. Context window: Depending on the model architecture, a context window of surrounding tokens may also be included in the input representation. This helps the model to capture the context in which each token appears.
4. Special tokens: Special tokens may be added to the input representation to indicate the beginning and end of the input sequence, as well as any other relevant information such as speaker identities or dialogue turn boundaries.
5. Preprocessing: Depending on the task and the specific requirements of the model, additional preprocessing steps may be applied to the input text. For example, stop words may be removed, and the text may be lowercased or lemmatized to reduce the dimensionality of the input space.

Overall, the specific approach to representing the input for a dialogue model will depend on the task, the available data, and the chosen model architecture.

## Dialogue History Modelling

To model multi-turn dialogue history, the authors choose to concatenate multiple utterances in the history into a long sequence, which is similar to single-turn dialogue history modelling (Table 2).

We can apply self-attention and we will attend to the different parts of this long sequence.

However, how we can attend to a specific turn of the dialogue?

We should be able to apply attention to the turns. For example, for the response that the network is currently generating, it might make sense to attend only to the previous 2 turns.

Maybe the topic has changed since the last 2 turns.

I was thinking the input sequence could be 1 utterance / 1 turn at a time, and we can stack the inputs in order to accumulate the whole dialogue history.

If we concatenate the whole dialogue history as an input for the LSTM the sequence could be quite long.

# 5. Baseline model - Implementation

### **5.1 Implementation Details:**

**5.2.1 Model - Encoder and Decoder**

Our baseline model is RNN-based and includes an encoder and a decoder with Bahdanau’s attention between them.

For the encoder we selected the GRU architecture. Both LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) can be used as encoders in a dialogue model, but we chose GRU because the network requires less memory, les computational ressources and has fewer parameters. It is less prone to overfitting compared to LSTM. It is also able to capture long term dependencies. For the single-turn dataset the average length of the dialogue utterances is 114 characters and the longest turn has 870 characters. So, the GRU network would be able to handle this task.

For the decoder we selected the AttentionalRNNDecoder, which is RNN decoder model with Bahdanau’s cross attention[1]. We also selected the GRU type of network.

We conducted the experiments with 2 layers for the encoder and 1 layer for the decoder.

During validation and testing, we use beamsearch to find the most likely output sequence based on the decoder output probabilities.

Tokenization

Character vs Word tokenization.

Character tokenization has the benefit of smaller vocabulary size. Also we have more flexibility in the construction of the words. However, the character sequences are longer than word sequences and they require more computational steps. For example, in [13] the authors report that the transformer model is able to exploit dependencies in over long distances, over 400 characters apart.

[**https://arxiv.org/pdf/1808.04444.pdf**](https://arxiv.org/pdf/1808.04444.pdf)

So, our model predicts one character (one token) at a time.

Also at training time, the model would process larger batch sizes (512) vs. the batch size of one that evaluation uses.

in its **inference/evaluation mode.** **That’s why it’s only processing one word at a time.**

# beam search

def beam\_search\_decoder(data, k):

sequences = [[list(), 0.0]]

# walk over each step in sequence

for row in data:

all\_candidates = list()

# expand each current candidate

for i in range(len(sequences)):

seq, score = sequences[i]

for j in range(len(row)):

print(j)

candidate = [seq + [j], score - np.log(row[j])]

all\_candidates.append(candidate)

# order all candidates by score

ordered = sorted(all\_candidates, key=lambda tup:tup[1])

# select k best

sequences = ordered[:k]

return sequences

**5.2.2 Hyper Parameters**

NOTE: The Neural Network model is usually high dimensional. The capacity

The vocabulary size is 120. We have reduced the vocabulary size by preprocessing the dataset and excluding the symbols that appear less than 100 times in the whole dataset. We apply character tokenizer for the baseline model in order to keep the vocabulary size low.

We employ 128 dimensional embeddings. Xavier initialization is used for the input connection weights.

[1] Reference: NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE, Bahdanau et.al. <https://arxiv.org/pdf/1409.0473.pdf>

We utilize Adaptive Sub-gradient Methods (AdaGrad) [Duchi et al., 2011] optimizer on mini-batches of size 32, with learning rate 0.15 and gradient clipping 2. The number of head is K = 5. As the number of heads increases, the semantic subspace of some heads become similar. 5 heads can attend most semantic parts in a query. We train our model in 300K iterations (about 10 epochs) and keep the best model on the validation set. For decoding process, we use beam search with a beam size of 5 and select the top-1 generated reply for evaluation. For the coefficient of penalty term, we take the hyper-parameters which

seq2seq model consisting of an encoder and a decoder

if self.step % self.hparams.gradient\_accumulation == 0:

**# gradient clipping & early stop if loss is not finite**

self.check\_gradients(loss)

self.optimizer.step()

self.optimizer.zero\_grad()

some combinations of batch size and learning rate yield better performance (faster convergence and higher accuracy) than others. Particularly in some cases, if you set wrong values for these 2 very important parameters, your model will never converge

parser.add\_argument("--**es\_patience**", type=int, default=5,

help="Early stopping's parameter: number of epochs with no improvement after which training will be stopped. Set to 0 to disable this technique.")

You can take a look at the wer.txt file as well to double-check the quality of the predicted output.

At the end of the training process, the sentence-error-rate (SER) of the model on the test set should be very low, ideally between 0% and 1%. A low SER indicates that the model can accurately generate output semantics, meaning that the task can be solved almost perfectly.

We employ the Adam optimizer (torch.optim.Adam) to update the model's parameters during training.

# 6. Transformer-based dialogue model

What are the advantages and disadvantages of the transformer model?

6.1 Self Attention

6.2 Hierarchical Self Attention

6.3 Beam Search vs Contrastive search

# Compared to Recurrent Neural Networks (RNNs), the transformer model has proven to be superior in quality for many sequence-to-sequence tasks while being more parallelizable. The nn.Transformer module relies entirely on an attention mechanism (implemented as [nn.MultiheadAttention](https://pytorch.org/docs/stable/generated/torch.nn.MultiheadAttention.html)) to draw global dependencies between input and output

During training, we use [nn.utils.clip\_grad\_norm\_](https://pytorch.org/docs/stable/generated/torch.nn.utils.clip_grad_norm_.html) to prevent gradients from exploding.

# Self-Attention - Dialogue History

<https://aclanthology.org/R19-1119.pdf>

The challenge of our chatbot is to capture the context of the conversation, to make it more human-like. The problem can be perceived as paying more attention to certain words while generating output.

<https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1645&context=etd_projects>

# Hierarchical Attention

We have to consider the hierarchy in data

<https://drive.google.com/drive/u/1/folders/17R5Wcq_4v6ZJiZ_982aNh-8SZdUiNmsp>

Who used hierarchical attention first?

Describe the article

Hierarchical self-attention is a type of attention mechanism used in natural language processing (NLP) tasks where the input consists of a hierarchical structure, such as paragraphs, sentences, and words. The goal of hierarchical self-attention is to enable the model to attend to different levels of abstraction within the input structure, and to capture both local and global dependencies.

Here's how you can implement hierarchical self-attention:

1. Prepare the input data: The input data should be structured hierarchically, such as a document consisting of paragraphs, which in turn consist of sentences, and so on. Each level of the hierarchy should be represented as a sequence of vectors (e.g., word embeddings).
2. Compute the initial embeddings: For each level of the hierarchy, compute the initial embeddings of the input vectors using a neural network. For example, you can use a convolutional neural network (CNN) or a recurrent neural network (RNN) to compute the embeddings of words within a sentence.
3. Compute the attention scores: Compute the attention scores between each pair of vectors within a level of the hierarchy using the dot-product attention mechanism or some other type of attention mechanism.
4. Compute the context vectors: Use the attention scores to compute the context vectors for each vector within a level of the hierarchy. The context vector for a vector is a weighted sum of all the vectors in the same level, where the weights are the attention scores.
5. Compute the attention scores across levels: Compute the attention scores between the context vectors of adjacent levels of the hierarchy, such as between the context vectors of sentences and paragraphs.
6. Compute the final context vectors: Use the attention scores across levels to compute the final context vectors for each vector in the lowest level of the hierarchy. The final context vector for a vector is a weighted sum of all the context vectors in higher levels of the hierarchy, where the weights are the attention scores across levels.
7. Use the final context vectors for downstream tasks: The final context vectors can be used as input to a downstream task, such as sentiment analysis or document classification.

Overall, implementing hierarchical self-attention involves using a multi-level attention mechanism to attend to different levels of abstraction within a hierarchical structure, and to capture both local and global dependencies.

## 

## 6.3 Contrastive Search

[Generating Human-level Text with Contrastive Search in Transformers 🤗 (huggingface.co)](https://huggingface.co/blog/introducing-csearch)

# 7. GPT2

7.1 Why we need Language Model

7.2 Why GPT2 and not BERT

GPT2 is a very large language model, trained on massive amounts of data. It’s architecture is Transformer Decoder

We use GPT2 here due to its large-scale pre-training corpus than other models and strong performance in other generation tasks

**GPT is autoregressive => it is designed to generate text is a sequential manner by predicting the next word in a sequence based on the preceding words**

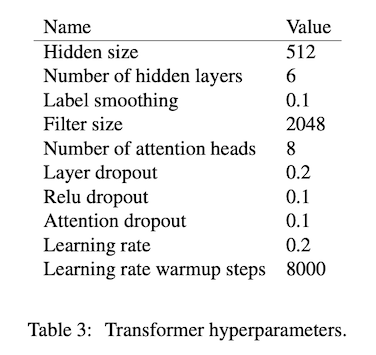
What is the advantage of having autoregressive model => the generation part

It can generate text

# 

# 8. Evaluation of dialogue models

https://tdk.bme.hu/VIK/DownloadPaper/A-Gutenberg-Dialogus-Adathalmaz-Neuralis

- loss

| **Parameters and Hyperparams** | **Value** |
| --- | --- |
| Hidden size | 512 |
| Number of layers | 3 |
| Label smoothing |  |
| Filter size |  |
| Number of attention heads | 8 |
| Layer dropout |  |
| Relu dropout |  |
| Learning rate |  |
| **Training** |  |
| Dataset size |  |
| Training time per epoch |  |

- WER, CER

- accuracy

- perplexity

- recall

5. Embedding metrics

- average

- extrema

- greedy

# 

# Evaluation metrics - Embedding measures

[EmbeddingBased/details.md at main · neural-dialogue-metrics/EmbeddingBased · GitHub](https://github.com/neural-dialogue-metrics/EmbeddingBased/blob/main/docs/details.md)

[EmbeddingBased/metrics.py at 3272d897c07ad2a2060aa45977ed3e84d578e63a · neural-dialogue-metrics/EmbeddingBased · GitHub](https://github.com/neural-dialogue-metrics/EmbeddingBased/blob/3272d897c07ad2a2060aa45977ed3e84d578e63a/embedding_based/metrics.py)

[**https://arxiv.org/pdf/1603.08023.pdf**](https://arxiv.org/pdf/1603.08023.pdf)

[**https://arxiv.org/pdf/1605.06069.pdf**](https://arxiv.org/pdf/1605.06069.pdf)

[**https://www.cs.toronto.edu/~lcharlin/papers/ubuntu\_dialogue\_dd17.pdf**](https://www.cs.toronto.edu/~lcharlin/papers/ubuntu_dialogue_dd17.pdf)

<https://www.ijcai.org/Proceedings/2018/0614.pdf>

3.3 Evaluation Researchers usually employ BLEU [Papineni et al., 2002] as an evaluation metric for generative dialogue systems. However, BLEU measures word overlap between the generated reply and the ground truth, which is too strict for evaluating dialogue systems due to significant diversity in the space of valid replies to a given context. Besides, [Liu et al., 2016; Tao et al., 2018] conduct empirical experiments and show weak correlation between BLEU and human annotation. In this paper, we consider three embedding-based metrics (including Embedding Average, Embedding Greedy and Embedding Extreme) to evaluate our model, following several recently studies on dialog systems [Serban et al., 2017; Xu et al., 2017]. The three metrics compute the similarity between the generated reply and reference reply according to the word embedding.

Embedding metrics. Embedding average, extrema, and greedy are widely used metrics [Liu et al., 2016, Serban et al., 2017b, Zhang et al., 2018c]. average measures the cosine similarity between the averages of word vectors of response and target utterances. extrema constructs a representation by taking the greatest absolute value for each dimension among the word vectors in the response and target utterances and measures the cosine similarity between them. Finally, greedy matches each response token to a target token (and vice versa) based on the cosine similarity between their embeddings and averages the total score across all words.

* Average Score, where the average\_score of all word vectors composing a sentence is taken into account as a summary of a sentence, and cosine similarity of these averages is used as the final score.

Average(sent) = sum(sent) / norm(sent)

AverageScore(source, target) = cosine\_similarity(Average(source), Average(target))

* Greedy Matching, as its name implies, it tries to find the maximum cosine similarity in a word-to-word basic, where each word of the source sentence is matched against all words of the target sentence to find the maximum cosine similarity. It then sums up these maximum cosine similarity scores for all words in a source sentence, normalized by the length of the source, following the pseudocode:

SumMaxCosine(source, target) = sum(max(cosine\_similarity(s, t) for s in source) for t in target) / len(source)

The same procedure is performed on the (target, source) pair and the final result is the average\_score of both, namely:

GreedyMacthing(source, target) = (SumMaxCosine(source, target) + SumMaxCosine(target, source)) / 2

* Vector Extrema, use a different way to generate a sentence representation from its constitute word embeddings. For each dimension of the word vectors, the extrema value is selected as below:

Min[i] = min(x for x in embeddings[i]) # Minimum of the i dimension.

Max[i] = max(x for x in embeddings[i]) # Maximum of the i dimension.

Extrema[i] = max(abs(Min[i]), Max[i]) # If the absolute of Min is large we take it. Otherwise take Max.

The sentence vector is then made up of these extrema values from all dimensions:

SentVec[i] = Extrema(i, embeddings)

Finally, the score is obtained by taking the cosine similarities of two SentVec.

**Automated Metrics**

We use **perplexity** on the **test data as the metric for intrinsic evaluation.** For extrinsic evaluation, we choose BLEU-2 and three types of word embedding similarities (Embedding Extrema, Embedding Average, Embedding Greedy) **to measure the closeness between a hypothesis and the corresponding ground-truth reference.**

## **Related Works**

These metrics are used as one of the metrics to evaluate the VHRED model proposed in Serban et al (2015 a), among others (the average length of response, the word entropy, and the utterance entropy w.r.t the unigram entropy of the training corpus and human evaluation). In *section 4.3 Results of Metric-based Evaluation*, settings of how the embedding-based metrics are used and their interpretation are detailed, along with evaluation results.

In the *How NOT to evaluate your dialogue system* paper, the authors discuss the embedding-based metrics in *section 3.2 Embedding-based Metrics*. They conclude that the embedding metrics, like other overlap-based metrics, do not correlate with human evaluation (not at all on Ubuntu Dialogue Corpus and weakly on Twitter Corpus). Despite this, Serban et al. interpret the metric as measuring *top similarity*. They then show that the HRED and VHRED models capture the topic in the context appropriately.

# How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation

# 

# Abstract We investigate evaluation metrics for dialogue response generation systems where supervised labels, such as task completion, are not available. Recent works in response generation have adopted metrics from machine translation to compare a model’s generated response to a single target response. We show that these metrics correlate very weakly with human judgements in the non-technical Twitter domain, and not at all in the technical Ubuntu domain. We provide quantitative and qualitative results highlighting specific weaknesses in existing metrics, and provide recommendations for future development of better automatic evaluation metrics for dialogue systems.

# 

# Similarity between response vectors is again computed using cosine distance. Intuitively, this approach prioritizes informative words over common ones; words that appear in similar contexts will be close together in the vector space. Thus, common words are pulled towards the origin because they occur in various contexts, while words carrying important semantic information will lie further away. By taking the extrema along each dimension, we are thus more likely to ignore common words.

# 

# 9. Inference

To perform inference with a Seq2Seq machine translation model, you can follow these steps:

1. Preprocessing: Preprocess the input text to be in the same format as the training data. This may involve tokenization, lowercasing, and other text normalization techniques.
2. Encoding: Convert the preprocessed input text into a fixed-length vector, which is the input to the decoder. This can be done using an encoder such as an LSTM or a bidirectional LSTM.
3. Decoding: Feed the encoded vector into the decoder, which is another LSTM or GRU. The decoder generates the output sequence one token at a time, starting with a special "start" token. At each step, the decoder generates the most likely next token based on its current state and the previously generated tokens.
4. Beam search: Beam search is used to generate the final output sequence. Beam search is a heuristic search algorithm that explores the most promising paths through the output space. It keeps track of a fixed number of hypotheses (called the "beam width"), and expands each hypothesis by generating the next token and adding it to the hypothesis.
5. Post-processing: Once the output sequence is generated, it may need to be post-processed to remove any special tokens (e.g. "start" and "end" tokens) and to convert it back into a human-readable format.

Overall, the process of performing inference with a Seq2Seq machine translation model can be quite complex, and there are many variations and optimizations that can be applied. However, the basic steps outlined above should provide a good starting point for implementing a simple system.

# 10. Results and Discussion

Training DNN models is often very time-consuming and expensive

FUTURE DIRECTIONS

6.4 Future Research Directions for End-to-End Systems Given the analysis performed in Section 4.6, we postulate several interesting directions for future research on end-to-end dialogue systems, particularly on the Ubuntu Dialogue Corpus. An important challenge in dialogue systems is the ability to understand the turn-taking structure of dialogue. This is a significant source of errors for the Dual Encoder model. Some progress in this direction has been made for end-to-end dialogue systems (Luan et al., 2016; Li et al., 2016a), using approaches derived from topic modelling or by explicitly modelling each user with a continuousvalued vector. However, this is still an open problem. This is related to the issue of end-to-end dialogue personalization, which involves building end-to-end dialogue systems that are tailored to a particular user and that evolve over time as the user’s preferences change. The largest source of errors from the analysis in Section 4.6 was in the failure to understand the semantic similarity between the context and response. This falls under the more general problem of natural language understanding, which arises in many NLP tasks. This will require adjustments in the architecture of end-to-end models to render them more suited to processing language. It is possible that insights can be derived from architectures developed on more targeted language understanding tasks, such as the CNN/ Daily Mail reading comprehension dataset (Hermann et al., 2015), where attention-based models have achieved strong performance. In order to be able to correctly answer questions regarding Ubuntu and solve the user’s problem, dialogue models will inevitably require some knowledge of the Ubuntu domain. This will most likely be achieved by using some source of external knowledge, in addition to the knowledge that is present in the dialogue of the Ubuntu Dialogue Corpus. Thus, an important direction for research is the investigation of methods that incorporate external knowledge sources with end-to-end dialogue systems. This applies more generally to any end-to-end system that is developed for the goaloriented setting, and may require imposing additional structure on the output space of the model. There is promising work in this direction from Wen et al. (2016), however methods must be derived that are effective in a larger and more general setting than restaurant recommendation. A common problem that has been observed when training generative end-to-end models that maximize the log-likelihood of the conversational response is that these models tend to produce generic responses at test time. This has been observed empirically (Vinyals and Le, 2015; Serban et al., 2016), and was also seen in some of the LSTM and HRED examples presented in Section

# 11. Conclusion

2. Which task is harder and why? Due to high diversity in DRG, DRG appears to be a harder problem because its decoder is less confident on what word to generate, and when to stop because of the uncertainty in attention and no length guidance from the source sentence. DRG’s attention is more smeared rather than focused like MT (which has a high degree of attention concentration). MT does much better on these, which enables a MT system to more confidently generate a translation.

3. What network internals have a major impact on the performance of each task? For different tasks, we have different answers. For MT, attention has a great impact on the final results. More focused attention means better translations. But for DRG, the answer is unclear. Its attention distribution does not have a strong correlation with the correct output. For both tasks, word frequency is a major factor that influences the performance.

4. What do we need to do in order to improve the performance of each task? For translation, since the degree of attention concentration has a clear correlation with the translation quality, it may be used as a translation quality measure. In designing new MT algorithms, we should try to improve the attention mechanism to enable it to have a higher degree of attention concentration. Furthermore, as more frequent words are more likely to be translated well, in data collection, one should focus on collecting more data containing those less frequent words. This is also the case for DRG. But overall, DRG appears to be a less well understood problem. The seq2seq model and attention may not be sufficient for the task.

## 12. References

[1] - model - ubuntu

Lowe, Ryan, Nissan Pow, Iulian Vlad Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. "Training end-to-end dialogue systems with the ubuntu dialogue corpus." Dialogue & Discourse 8, no. 1 (2017): 31-65.

[2] - dataset - ubuntu

Lowe, Ryan, Nissan Pow, Iulian Serban, and Joelle Pineau. "The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems." arXiv preprint arXiv:1506.08909 (2015).

[3] - MovieTriples dataset - generative model- Hierarchical Recurrent Encoder-Decoder

Serban, Iulian, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. "Building end-to-end dialogue systems using generative hierarchical neural network models." In Proceedings of the AAAI conference on artificial intelligence, vol. 30, no. 1. 2016.

[4] - eval – embedding metrics

Serban, Iulian, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron Courville, and Yoshua Bengio. "A hierarchical latent variable encoder-decoder model for generating dialogues." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31, no. 1. 2017.

[5] - model – ubuntu and twitter – generative - LSTM, Hierarchical Recurrent Encoder-Decoder

Serban, Iulian, Tim Klinger, Gerald Tesauro, Kartik Talamadupula, Bowen Zhou, Yoshua Bengio, and Aaron Courville. "Multiresolution recurrent neural networks: An application to dialogue response generation." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31, no. 1. 2017.

[6] - model - MultiWoz - objective function – Semantic Loss - generative - LSTM

Parthasarathi, Prasanna, Mohamed Abdelsalam, Joelle Pineau, and Sarath Chandar. "A Brief Study on the Effects of Training Generative Dialogue Models with a Semantic loss." arXiv preprint arXiv:2106.10619 (2021).

[7] - model

Zhang, Yizhe, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. "Dialogpt: Large-scale generative pre-training for conversational response generation." arXiv preprint arXiv:1911.00536 (2019).

[8] - attention

Adeniji, Ademi, Nate Lee, and Vincent Liu. "Sequence-to-Sequence Generative Argumentative Dialogue Systems with Self-Attention." (2019).

[9] - eval

Yeh, Yi-Ting, Maxine Eskenazi, and Shikib Mehri. "A comprehensive assessment of dialog evaluation metrics." arXiv preprint arXiv:2106.03706 (2021).

[10] - eval

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

[11]-eval

Ryan Lowe\*, Michael Noseworthy\*, Iulian Serban, Nicholas Angelard-Gontier, Yoshua Bengio, Joelle Pineau. "Towards an Automatic Turing Test: Learning to Evaluate Dialogue Responses."

In Association for Computational Linguistics (ACL), 2017.

[12]-eval

Lizi Liao, Le Hong Long, Yunshan Ma, Wenqiang Lei, and Tat-Seng Chua. 2021. Dialogue State Tracking with Incremental Reasoning. Transactions of the Association for Computational Linguistics, 9:557–569.