Neural Network Architecture for the General Conversation Challenge

Andrey Kulagin
Department of Computer Science
Innopolis University
Innopolis, Russia
a.kulagin@innopolis.ru

***Abstract*—This paper contains a survey of various neural network architectures for the General Conversation Challenge - competition organized by Yandex where a developed system have to choose a suitable phrase for a voice assistant based on some previous context (several utterances).**

***Keywords*—dialogue systems, RNN, conversation challenge**

I. Introduction

Building an open-domain conversational assistant that is clever and fun is one the most exciting applications and research frontiers of artificial intelligence nowadays. The core and very challenging task of such an assistant is providing coherent and entertaining replies to the users at all steps of their conversation with the assistant [1]

There are two major types of approaches for response generation in dialog systems: generative and retrieval-based. The approaches of the first kind build complex language models to generate replies given the context of the conversation, word by word. The approaches of the second kind assume that "everything that needs to be said has already been said" and, instead of generation of responses, search for the best response for a given situation in a large collection of candidate responses.

In the April of 2018 Yandex organized a competition called General Conversation Challenge [2] which is focused on improving response selection mechanism of the retrieval-based conversational assistants.

II. Challenge description

A. Dataset

The initial dataset which is used for the challenge is the OpenSubtitles [3] dataset, and particularly its Russian subset. It contains an ordered set of utterances. In most cases, each utterance is a reply to the previous utterance in a dialogue of two characters in a movie. Organizers randomly sampled short episodes of these dialogues for the training examples.

Each conversation episode always consists of two parts: "Context" and "Reply". For example:

- **context_2**: Character A asks a question
- **context_1**: Character B replies to that question
- **context_0**: Character A asks another question
- **reply**: Character B replies to the second question

"Context" part contains three utterances in 50% of cases, two in 25%, and one in 25%. "Reply" utterance always follows "Context" (so, concludes the episode). The goal of the challenge is to find most appropriate and interesting replies among all reply candidates (up to 6) provided to the participants for the given "Context".

All candidate replies are labeled by crowdworkers from Yandex.Toloka (an analogue of Amazon Turk) using the following quality labelling guidelines.

- **Good (1)**: the reply is appropriate (meaningful given the context) and interesting (non-trivial/entertaining, context-specific and potentially motivates the user to continue the conversation)
- **Neutral (0)**: the reply is appropriate (makes sense given the context), but not very interesting (trivial/boring, not very context-specific and rather makes the user want to end the conversation)
- **Bad (-1)**: the reply makes no sense given the context

Moreover for each label in the training data we have the confidence measure (a number in the interval [0, 1]) that indicates how certain were the crowdworkers about the label they assigned to the candidate reply.

B. Task

The task is to return a ranking of the candidate replies for a given context in the decreasing order of their predicted scores. The metric used to evaluate the rankings is NDCG (Normalized DCG) [4]:

$$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)}$$

$$nDCG_p = \frac{DCG_p}{IDCG_p}$$

IDCG is measured after ranking all candidates in the decreasing order of their actual (not predicted) labels.

rel_i takes three values - 1, 0 and -1 - corresponding to labels **good**, **neutral** and **bad**

The final score is the average value of NDCG for all contexts multiplied by 100000.

III. Word representation, loss function and first architecture

When dealing with NLP tasks the first thing to consider is usually how to represent words and sentences... Currently the best results for word representations for text classification, generation tasks demonstrate so called *word embeddings* or distributed word representations. The main idea is to map each word into a vector of real values from $\mathbb{R}^d$ where d is usually about several hundreds. There are different methods with such idea:

- word2vec [5]
- GloVe [6]
- FastText [7]

We'll be using FastText because it works on a subword level and provides better results for languages with rich morphology (such as Russian). We don't have a large corpus to train word representations by ourselves but fortunately we don't have to do it: there are pretrained models on the FastText website [8] for 157 languages (including Russian), trained on Common Crawl and Wikipedia.

So after the first step of our pipeline each utterance will be converted to a sequence of real-valued vectors with 300 dimensions each.

Then we want to represent the *"meaning"* of the whole phrase. This is usually done via Recurrent Neural Network architectures such as LSTM [9] or GRU [10], which accept a sequence of vectors (representations of our words), run through it, maintaining and updating internal state and returns a vector (for example with 100 dimensions), describing the whole sequence.

We have maximum 4 utterances (*context_2*, *context_1*, *context_0* and *reply)*. And the task is to find out how suitable *reply* for the given contexts. Our first architecture will combine the meanings of all utterances (each is a 100-dimensional vector as described above) via 1 Dense layer with 200 neurons and ReLU activation function.

But what we expect as an output of our model? What should we predict and how to measure error (what loss function to use)?

Target loss function, which is used for ranking participants' solutions is nDCG (let's call it **yandex_score**, the larger it, the better). But we cannot use it directly for our model because it is non-differentiable loss function. Moreover we are dealing with data where we have not ranking of results but classification: Good (1), Neutral (0), Bad (-1), it is a bit different nature.

So the first idea is to convert the task into classification problem, where the model must predict labels (Good, Neutral, Bad). We can use the last layer with 3 neurons (for 3 labels) and softmax activation, loss will be categorical crossentropy. This approach gives **yandex_score=84800** (with the baseline architecture).

But our labels are not independent, in fact we have the order Bad $<$ Neutral $<$ Good. So, probably this task is more about regression than classification. Also to rank 6 replies for the given context for the test data it will be helpful to have real-valued scores and not just labels. So let's think of it as a regression problem where we have to predict score for a particular reply. For example, if we have a train sample with contexts and reply which is marked as Good, target (true) value will be $y_t = 1$. So we will have only one neuron as the last layer of our network. Loss function will be mean squared error. This is the architecture we will refer to as baseline. Figure 1 shows it.

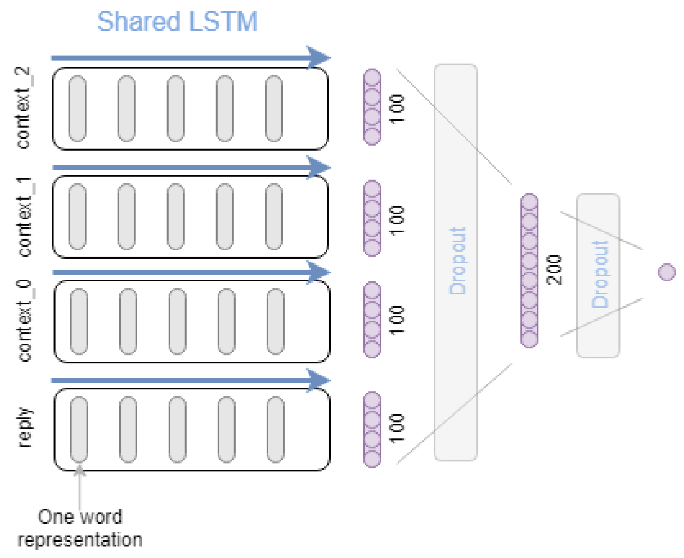

Fig. 1. Baseline architecture

We will use Keras on top of TensorFlow to code our model. Some techinal details:

1) Adam optimizer [11] is used to improve gradient descent
2) Because we are dealing with Recurrent Neural Network it is important to clip gradients to prevent them from exploding. All parameter gradients are clipped to a maximum norm of 1.
3) Dropout is used between dense layers to prevent model from early overfitting [12]
4) A small fraction (0.1) of the train dataset is used for validation and Early Stopping (also to prevent overfitting)
5) Confidence information (number from [0,1]) is used as a sample's weight

Best validation_loss for the baseline architecture is 0.671 and **yandex_score=85600**

IV. New loss, Bidirectional GRU and more neurons

In this section will be described several improvements which together give **yandex_score=86300**.

The first one is the new loss function. Previously we used mean squared error, probably best known loss for regression. Its plot represented on figure 2

But if for example $y_t = -1$ (Bad reply) and the model predicted -1.5, do we really have to penalize it? Intuition says that for "Bad" samples everything below -1 is ok, also for

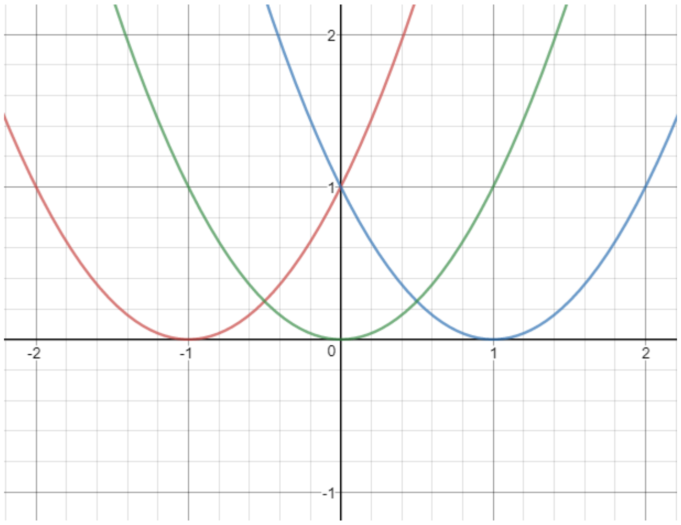

Fig. 2. Mean Squared Error: Red for $y_t = -1$, Green for $y_t = 0$ and Blue for $y_t = 1$

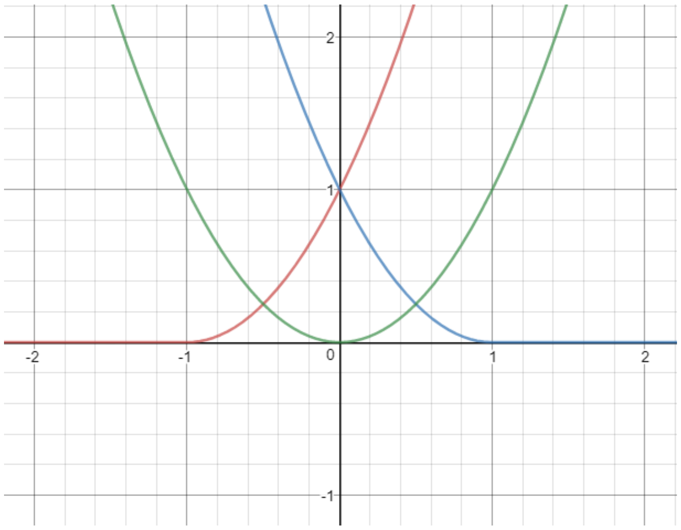

Fig. 3. New Loss function: Red for $y_t = -1$, Green for $y_t = 0$ and Blue for $y_t = 1$

"Good" ones everything greater 1 too. So our new loss function will be:

$$Loss(y) = \begin{cases} (y - y_t)^2 & y \geq -1, y_t = -1 \\ 0 & y < -1, y_t = -1 \\ (y - y_t)^2 & y_t = 0 \\ (y - y_t)^2 & y \leq 1, y_t = 1 \\ 0 & y > 1, y_t = 1 \end{cases}$$

You can see its plot on figure 3.

With new loss function model achieves **yandex_score=85800**

The next change will be usage of GRU instead of LSTM. GRU has almost two times less parameters to learn comparing to LSTM and for short phrases this architecture should perform a bit better than LSTM (we are not dealing with "War and Peace", just short utterances). Improvement was not so big but the model achieved **yandex_score=85900**

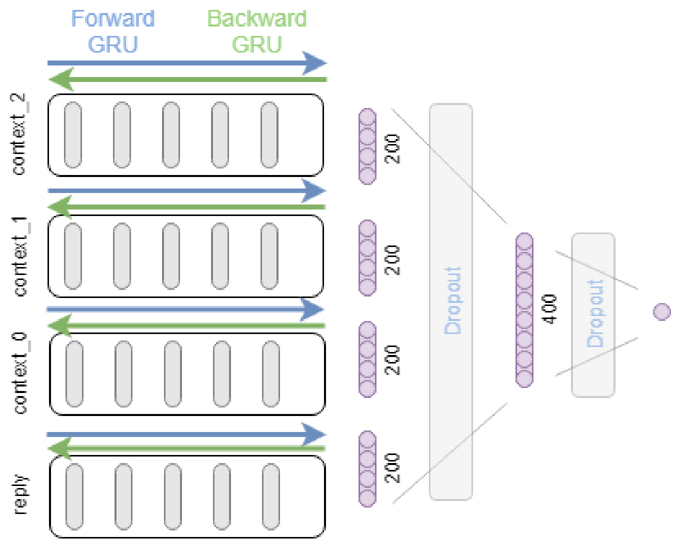

Fig. 4. Final architecture

TABLE I. COMPARISON OF DIFFERENT APPROACHES

Architecture, improvements	parameters	val_loss	yandex_score
Baseline (regression)	240801	0.6710	85600
new loss	240801	0.6655	85800
new loss, GRU	200701	0.6630	85900
new loss, Bidirectional GRU	401001	0.6610	86100
new loss, Bidirectional GRU, Dense(400)	561401	0.6570	86300

Also we can run GRU not only from the first word to the last, but also backwards and then concatenate the results. So we will have $100 * 2 = 200$ dimension for vector per phrase. This change results in **yandex_score=86100**.

And the last architecture improvement will be increasing the number of neurons from 200 to 400 in the Dense layer to allow the model capture more hidden details. The final score as already mentioned is **yandex_score=86300**. You can find the final architecture on figure 4.

V. CONCLUSION

Having started with the baseline architecture step by step we've been improving our model: new loss function, different recurrent unit, bidirectional pass, more neurons... The final version achieved **yandex_score=86300** and the **12-th place** out of **200** competitors on the General Conversation Challenge (public part). The source code of models can be found here [13].

REFERENCES

[1] A Survey on Dialogue Systems: Recent Advances and New Frontiers. KDD 2017 http://www.kdd.org/exploration_files/19-2-Article3.pdf

[2] General Conversation Challenge https://contest.yandex.ru/algorithm2018/contest/7914/problems/

[3] Open Subtitles http://opus.nlpl.eu/download.php?f=OpenSubtitles2016/en-ru.txt.zip

[4] Discounted Cumulative Gain https://en.wikipedia.org/wiki/Discounted_cumulative_gain

[5] Mikolov, Tomas, et al. ”Distributed representations of words and phrases and their compositionality.” Advances in neural information processing systems. 2013.

[6] Pennington, Jeffrey, Richard Socher, and Christopher Manning. ”Glove: Global vectors for word representation.” Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

[7] Joulin, Armand, et al. ”Bag of tricks for efficient text classification.” arXiv preprint arXiv:1607.01759 (2016).

[8] FastText https://fasttext.cc/

[9] Long Short-Term Memory https://en.wikipedia.org/wiki/Long_short-term_memory

[10] Gated recurrent unit https://en.wikipedia.org/wiki/Gated_recurrent_unit

[11] Kingma, Diederik P., and Jimmy Ba. ”Adam: A method for stochastic optimization.” arXiv preprint arXiv:1412.6980 (2014).

[12] Srivastava, Nitish, et al. ”Dropout: A simple way to prevent neural networks from overfitting.” The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

[13] https://github.com/and-kul/yandex_algorithm_2018