LEARNING GOAL-ORIENTED VISUAL DIALOG VIA TEMPERED POLICY GRADIENT

Rui Zhao, Volker Tresp

Ludwig Maximilian University, Oettingenstr. 67, 80538 Munich, Germany

Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 Munich, Germany

ABSTRACT

Learning goal-oriented dialogues by means of deep reinforcement learning has recently become a popular research topic. However, commonly used policy-based dialogue agents often end up focusing on simple utterances and suboptimal policies. To mitigate this problem, we propose a class of novel temperature-based extensions for policy gradient methods, which are referred to as Tempered Policy Gradients (TPGs). On a recent AI-testbed, i.e., the GuessWhat?! game, we achieve significant improvements with two innovations. The first one is an extension of the state-of-the-art solutions with Seq2Seq and Memory Network structures that leads to an improvement of 7%. The second one is the application of our newly developed TPG methods, which improves the performance additionally by around 5% and, even more importantly, helps produce more convincing utterances.

***Index Terms*—** Goal-Oriented Dialog System, Deep Reinforcement Learning, Recurrent Neural Network

1. INTRODUCTION

In recent years, deep learning has shown convincing performance in various areas such as image recognition, speech recognition, and natural language processing (NLP). Deep neural nets are capable of learning complex dependencies from huge amounts of data and its human generated annotations in a supervised way. In contrast, reinforcement learning agents [2] can learn directly from their interactions with the environment without any supervision and surpass human performance in several domains, for instance in the game of GO [3], as well as many computer games [4]. In this paper we are concerned with the application of both approaches to goal-oriented dialogue systems [5, 6, 7, 8, 9, 10, 11, 7, 12, 13, 14], a problem that has recently caught the attention of machine learning researchers. De Vries et al. [6] have proposed as AI-testbed a visual grounded object guessing game called GuessWhat?!. Das et al. [7] formulated a visual dialogue system which is about two chatbots asking and answering questions to identify a specific image within a group of images. More practically, dialogue agents have been applied to negotiate a deal [12] and access certain information from knowledge bases [13]. The essential idea in these systems is to train different dialogue agents to accomplish the tasks. In those papers, the agents have been trained with policy gradients, i.e. REINFORCE [15].

In order to improve the exploration quality of policy gradients, we present three instances of temperature-based methods. The first one is a single-temperature approach which is very easy to apply. The second one is a parallel approach with multiple temperature policies running concurrently. This second approach is more demanding on computational resources, but results in more stable solutions. The third one is a temperature policy approach that dynamically adjusts the temperature for each action at each time-step, based on action frequencies. This dynamic method is more sophisticated and proves more efficient in the experiments. In the experiments, all these methods demonstrate better exploration strategies in comparison to the plain policy gradient.

We demonstrate our approaches using a real-world dataset called GuessWhat?!. The GuessWhat?! game [6] is a visual object discovery game between two players, the Oracle and the Questioner. The Questioner tries to identify an object by asking the Oracle questions. The original works [6, 8] first proposed supervised learning to simulate and optimize the game. Strub et al. [8] showed that the performance could be improved by applying plain policy gradient reinforcement learning, which maximizes the game success rate, as a second processing step. Building on these previous works, we propose two network architecture extensions. We utilize a Seq2Seq model [16] to process the image along with the historical dialogues for question generation. For the guessing task, we develop a Memory Network [17] with Attention Mechanism [18] to process the generated question-answer pairs. We first train these two models using the plain policy gradient and use them as our baselines. Subsequently, we train the models with our new TPG methods and compare the performances with the baselines. We show that the TPG method is compatible with state-of-the-art architectures such as Seq2Seq and Memory Networks and contributes orthogonally to these advanced neural architectures. To the best of our knowledge, the presented work is the first to propose temperature-based policy gradient methods to leverage explo-

This paper is an extended version of the IJCAI workshop paper [1].

ration and exploitation in the field of goal-oriented dialogue systems. We demonstrate the superior performance of our TPG methods by applying it to the GuessWhat?! game.

2. PRELIMINARIES

In our notation, we use $\mathbf{x}$ to denote the input to a policy network π, and x_i to denote the i-th element of the input vector. Similarly, $\mathbf{w}$ denotes the weight vector of π, and w_i denotes the i-th element of the weight vector of that π. The output y is a multinoulli random variable with N states that follows a probability mass function, $f(y = n \mid \pi(\mathbf{x} \mid \mathbf{w}))$, where $\Sigma_{n=1}^{N} f(y = n \mid \pi(\mathbf{x} \mid \mathbf{w})) = 1$ and $f(\cdot) \geq 0$. In a nutshell, a policy network parametrizes a probabilistic unit that produces the sampled output, mathematically, $y \sim f(\pi(\mathbf{x} \mid \mathbf{w}))$.

Typically, the expected value of the accumulated reward, i.e. return, conditioned on the policy network parameters $E(r \mid \mathbf{w})$ is used. Here, E denotes the expectation operator, r the accumulated reward signal, and $\mathbf{w}$ the network weight vector. The objective of reinforcement learning is to update the weights in a way that maximizes the expected return at each trial. In particular, the REINFORCE updating rule is: $\Delta w_i = \alpha_i(r - b_i)e_i$, where Δw_i denotes the weight adjustment of weight w_i, α_i is a nonnegative learning rate factor, and b_i is a reinforcement baseline. The e_i is the *characteristic eligibility* of w_i, defined as $e_i = (\partial f/\partial w_i)/f = \partial \ln f/\partial w_i$. Williams [15] has proved that the updating quantity, $(r - b_i)\partial \ln f/\partial w_i$, represents an unbiased estimate of $\partial E(r \mid \mathbf{w})/\partial w_i$.

3. TEMPERED POLICY GRADIENT

In order to improve the exploration quality of REINFORCE in the task of optimizing policy-based dialogue agents, we attempt to find the optimal compromise between exploration and exploitation. In TPGs we introduce a parameter τ, the sampling temperature of the probabilistic output unit, which allows us to explicitly control the strengths of the exploration.

3.1. Exploration and Exploitation

The trade-off between exploration and exploitation is one of the great challenges in reinforcement learning [2]. To obtain a high reward, an agent must exploit the actions that have already proved effective in getting more rewards. However, to discover such actions, the agent must try actions, which appear suboptimal, to explore the action space. In a stochastic task like text generation, each action, i.e. a word, must be tried many times to find out whether it is a reliable choice or not. The exploration-exploitation dilemma has been intensively studied over many decades [19, 20, 21]. Finding the balance between exploration and exploitation is considered crucial for the success of reinforcement learning [22].

3.2. Temperature Sampling

In text generation, it is well-known that the simple trick of temperature adjustment is sufficient to shift the language model to be more conservative or more diversified [23]. In order to control the trade-off between exploration and exploitation, we borrow the strength of the temperature parameter $\tau \geq 0$ to control the sampling. The output probability of each word is transformed by a temperature function as:

$$f^{\tau}(y = n \mid \pi(\mathbf{x} \mid \mathbf{w})) = \frac{f(y = n \mid \pi(\mathbf{x} \mid \mathbf{w}))^{\frac{1}{\tau}}}{\Sigma_{m=1}^{N} f(y = m \mid \pi(\mathbf{x} \mid \mathbf{w}))^{\frac{1}{\tau}}}.$$

We use notation f^{τ} to denote a probability mass function f that is transferred by a temperature function with temperature τ. When the temperature is high, $\tau > 1$, the distribution becomes more uniform; when the temperature is low, $\tau < 1$, the distribution appears more spiky.

3.3. Tempered Policy Gradient Methods

Here, we introduce three instances of TPGs in the domain of goal-oriented dialogues, including single, parallel, and dynamic tempered policy gradient methods.

Single-TPG: The Single-TPG method simply uses a global temperature τ_{global} during the whole training process, i.e., we use $\tau_{global} > 1$ to encourage exploration. The forward pass is represented mathematically as: $y^{\tau_{global}} \sim f^{\tau_{global}}(\pi(\mathbf{x} \mid \mathbf{w}))$, where $\pi(\mathbf{x} \mid \mathbf{w})$ represents a policy neural network that parametrizes a distribution $f^{\tau_{global}}$ over the vocabulary, and $y^{\tau_{global}}$ means the word sampled from this tempered word distribution. After sampling, the weight of the neural net is updated using,

$$\Delta w_i = \alpha_i(r - b_i)\partial \ln f(y^{\tau_{global}} \mid \pi(\mathbf{x} \mid \mathbf{w}))/\partial w_i.$$

Noteworthy is that the actual gradient, $\partial \ln f(y^{\tau_{global}} \mid \pi(\mathbf{x} \mid \mathbf{w}))/\partial w_i$, depends on the sampled word, $y^{\tau_{global}}$, however, does not depend directly on the temperature, τ. With Single-TPG and $\tau > 1$, the entire vocabulary of a dialogue agent is explored more efficiently than by REINFORCE, because non-preferred words have a higher probability of being explored.

Parallel-TPG: A more advanced version of Single-TPG is the Parallel-TPG that deploys several Single-TPGs concurrently with different temperatures, $\tau_1, ..., \tau_n$, and updates the weights based on all generated samples. During the forward pass, multiple copies of the neural nets parameterize multiple word distributions. The words are sampled in parallel at various temperatures, mathematically, $y^{\tau_1}, ..., y^{\tau_n} \sim f^{\tau_1,...,\tau_n}(\pi(\mathbf{x} \mid \mathbf{w}))$. After sampling, in the backward pass the weights are updated with the sum of gradients. The formula is given by

$$\Delta w_i = \Sigma_k \alpha_i(r - b_i)\partial \ln f(y^{\tau_k} \mid \pi(\mathbf{x} \mid \mathbf{w}))/\partial w_i,$$

where $k \in \{1, ..., n\}$. The combinational use of higher and lower temperatures ensures both exploration and exploitation

at the same time. The sum over weight updates of parallel policies gives a more accurate Monte Carlo estimate of $\partial E(r \mid \mathbf{w})/\partial w_i$, due to the nature of Monte Carlo methods [24]. Thus, compared to Single-TPG, we would argue that Parallel-TPG is more robust and stable, although Parallel-TPG needs more computational power. However, these computations can easily be distributed in a parallel fashion using state-of-the-art graphics processing units.

Dynamic-TPG: As a third variant, we introduce the Dynamic-TPG, which is the most sophisticated approach in the current TPG family. The essential idea is that we use a heuristic function h to assign the temperature τ to the word distribution at each time step, t. The temperature is bounded in a predefined range $[\tau_{min}, \tau_{max}]$. The heuristic function we used here is based upon the term frequency inverse document frequency, *tf-idf* [25]. In the context of goal-oriented dialogues, we use the counted number of each word as term frequency *tf* and the total number of generated dialogues during training as document frequency *df*. We use the word that has the highest probability to be sampled at current time-step, y_t^*, as the input to the heuristic function h. Here, y_t^* is the maximizer of the probability mass function f. Mathematically, it is defined as $y_t^* = \text{argmax}(f(\pi(\mathbf{x} \mid \mathbf{w})))$. We propose that *tf-idf*(y_t^*) approximates the concentration level of the distribution, which means that if the same word is always sampled from a distribution, then the distribution is very concentrated. Too much concentration prevents the model from exploration, so that a higher temperature is needed. In order to achieve this effect, the heuristic function is defined as

$$\begin{aligned}\tau_t^h &= h(\textit{tf-idf}(y_t^*)) \\ &= \tau_{min} + (\tau_{max} - \tau_{min}) \frac{\textit{tf-idf}(y_t^*) - \textit{tf-idf}_{min}}{\textit{tf-idf}_{max} - \textit{tf-idf}_{min}}.\end{aligned}$$

With this heuristic, words that occur very often are depressed by applying a higher temperature to those words, making them less likely to be selected in the near future. In the forward pass, a word is sampled using $y^{\tau_t^h} \sim f^{\tau_t^h}(\pi(\mathbf{x} \mid \mathbf{w}))$. In the backward pass, the weights are updated correspondingly:

$$\Delta w_i = \alpha_i (r - b_i) \partial \ln f(y^{\tau_t^h} \mid \pi(\mathbf{x} \mid \mathbf{w}))/\partial w_i,$$

where τ_t^h is the temperature calculated from the heuristic function. Compared to Parallel-TPG, the advantage of Dynamic-TPG is that it assigns temperature more appropriately, without increasing the computational load.

4. GUESSWHAT?! GAME

We evaluate our methods using a recent testbed for AI, called the GuessWhat?! game [6], available at `https://guesswhat.ai`. The dataset consists of 155 k dialogues, including 822 k question-answer pairs, each composed of around 5 k words, about 67 k images [26] and 134 k objects. The game is about visual object discovery trough a multi-round QA among different players.

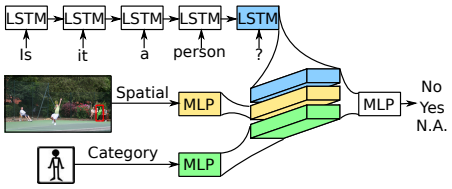


Fig. 1: Oracle model

Formally, a GuessWhat?! game is represented by a tuple (I, D, O, o^*), where $I \in \mathbb{R}^{H \times W}$ denotes an image of height H and width W; D represents a dialogue composed of M rounds of question-answer pairs (QAs), $D = (\mathbf{q}_m, a_m)_{m=1}^M$; O stands for a list of K objects $O = (o_k)_{k=1}^K$; and o^* is the target object. Each question is a sequence of words, $\mathbf{q}_m = \{y_{m,1}, ..., y_{m,N_m}\}$ with length N_m. The words are taken from a defined vocabulary V, which consists of the words and a start token and an end token. Each answer is either yes, no, or not applicable, i.e. $a_m \in \{yes, no, n.a.\}$. For each object o_k, there is a corresponding object category $c_k \in \{1, ..., C\}$ and a pixel-wise segmentation mask $S_k \in \{0, 1\}^{H \times W}$. Finally, we use colon notation (:) to select a subset of a sequence, for instance, $(\mathbf{q}, a)_{1:m}$ refers to the first m rounds of QAs in a dialogue.

4.1. Models and Pretraining

Following [8], we first train all three models in a supervised fashion.

Oracle: The task of the Oracle is to answer questions regarding to the target object. We outline here the simple neural network architecture that achieved the best performance in the study of [6], and which we also used in our experiments. The input information used here is of three modalities, namely the question $\mathbf{q}$, the spatial information $x_{spatial}^*$ and the category c^* of the target object. For encoding the question, de Vries et al. first use a lookup table to learn the embedding of words, then use a one layer long-short-term-memory (LSTM) [27] to encode the whole question. For spatial information, de Vries et al. extract an 8-dimensional vector of the location of the bounding box $[x_{min},\ y_{min},\ x_{max},\ y_{max},\ x_{center},\ y_{center},\ w_{box},\ h_{box}]$, where x, y denote the coordinates and w_{box}, h_{box} denote the width and height of the bounding box, respectively. De Vries et al. normalize the image width and height so that the coordinates range from -1 to 1. The origin is at the image center. The category embedding of the object is also learned with a lookup table during training. At the last step, de Vries et al. concatenate all three embeddings into one feature vector and fed it into a one hidden layer multilayer perceptron (MLP). The softmax output layer predicts the distribution, $\text{Oracle} := p(a \mid \mathbf{q},\ c^*,\ x_{spatial}^*)$, over the three classes,

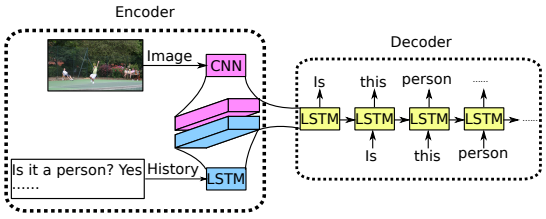


Fig. 2: Question-Generator model

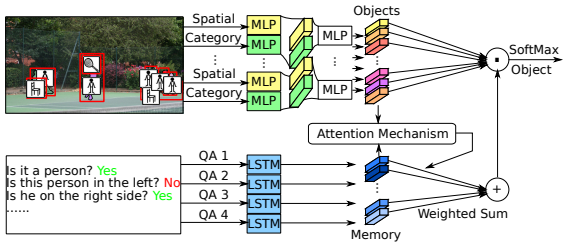


Fig. 3: Guesser model

including no, yes, and not applicable. The model is trained using the negative log-likelihood criterion. The Oracle structure is shown in Fig. 1.

Question-Generator: The goal of the Question-Generator (QGen) is to ask the Oracle meaningful questions, $\mathbf{q}_{m+1}$, given the whole image, I, and the historical question-answer pairs, $(\mathbf{q}, a)_{1:m}$. In previous work [8], the state transition function was modelled as an LSTM, which was trained using whole dialogues so that the model memorizes the historical QAs. We refer to this as dialogue level training. We develop a novel QGen architecture using a modified version of the Seq2Seq model [16]. The modified Seq2Seq model enables *question level training*, which means that the model is fed with historical QAs, and then learns to produce a new question. Following [8], we first encode the whole image into a fixed-size feature vector using the VGG-net [28]. The features come from the fc-8 layer of the VGG-net. For processing historical QAs, we use a lookup table to learn the word embeddings, then again use an LSTM encoder to encode the history information into a fixed-size latent representation, and concatenate it with the image representation:

$$\mathbf{s}^{enc}_{m,Nm} = \text{encoder}((\text{LSTM}(\mathbf{q}, a)_{1:m}), \text{VGG}(I)).$$

The encoder and decoder are coupled by initializing the decoder state with the last encoder state, mathematically, $\mathbf{s}^{dec}_{m+1,0} = \mathbf{s}^{enc}_{m,Nm}$. The LSTM decoder generates each word based on the concatenated representation and the previous generated word (note the first word is a start token):

$$y_{m+1,n} = \text{decoder}(\text{LSTM}((y_{m+1,n-1}, \mathbf{s}^{dec}_{m+1,n-1})).$$

The decoder shares the same lookup table weights as the encoder. The Seq2Seq model, consisting of the encoder and the decoder, is trained end-to-end to minimize the negative log-likelihood cost. During testing, the decoder gets a start token and the representation from the encoder, and then generates each word at each time step until it encounters a question mark token, $\text{QGen} := p(y_{m+1,n} \mid (\mathbf{q}, a)_{1:m}, I)$. The output is a complete question. After several question-answer rounds, the QGen outputs an end-of-dialogue token, and stops asking questions. The overall structure of the QGen model is illustrated in Fig. 2.

Guesser: The goal of the Guesser model is to find out which object the Oracle model is referring to, given the complete history of the dialogue and a list of objects in the image, $p(o^* \mid (\mathbf{q}, a)_{1:M}, x^O_{spatial}, c^O)$. The Guesser model has access to the spatial, $x^O_{spatial}$, and category information, c^O, of the objects in the list. The task of the Guesser model is challenging because it needs to understand the dialogue and to focus on the important content, and then guess the object. To accomplish this task, we decided to integrate the Memory [17] and Attention [18] modules into the Guesser architecture used in the previous work [8]. First, we use an LSTM header to process the varying lengths of question-answer pairs in parallel into multiple fixed-size vectors. Here, each QA-pair has been encoded into some facts, $\text{Fact}_m = \text{LSTM}((\mathbf{q}, a)_m)$, and stored into a memory base. Later, we use the sum of the spatial and category embeddings of all objects as a key, $\text{Key}_1 = \text{MLP}(x^O_{spatial}, c^O)$, to query the memory and calculate an attention mask, $\text{Attention}_1(\text{Fact}_m) = \text{Fact}_m \odot \text{key}_1$, over each fact. Next, we use the sum of attended facts and the first key to calculate the second key. Further, we use the second key to query the memory base again to have a more accurate attention. These are the so called "two-hops" of attention in the literature [17]. Finally, we compare the attended facts with each object embedding in the list using a dot product. The most similar object to these facts is the prediction, $\text{Guesser} := p(o^* \mid (\mathbf{q}, a)_{1:M}, x^O_{spatial}, c^O)$. The intention of using the attention module here is to find out the most relevant descriptions or facts concerning the candidate objects. We train the whole Guesser network end-to-end using the negative log-likelihood criterion. A more graphical description of the Guesser model is shown in Fig. 3.

4.2. Reinforcement Learning

Now, we post-train the QGen and the Guesser model with reinforcement learning. We keep the Oracle model fixed. In each game episode, when the models find the correct object, $r = 1$, otherwise, $r = 0$.

Next, we can assign credits for each action of the QGen and the Guesser models. In the case of the QGen model, we spread the reward uniformly over the sequence of actions in the episode. The baseline function, b, used here is the running average of the game success rate. Consider that the Guesser model has only one action in each episode, i.e., taking the guess. If the Guesser finds the correct object, then it gets an

immediate reward and the Guesser's parameters are updated using the REINFORCE rule without baseline. The QGen is trained using the following four methods.

REINFORCE: The baseline method used here is REINFORCE [15]. During training, in the forward pass the words are sampled with $\tau = 1$, $y_{m+1,n} \sim f(\mathrm{QGen}(\mathbf{x} \mid \mathbf{w}))$. In the backward pass, the weights are updated using REINFORCE, $\mathbf{w} = \mathbf{w} + \alpha(r - b)\nabla_{\mathbf{w}} \ln f(y_{m+1,n} \mid \mathrm{QGen}(\mathbf{x} \mid \mathbf{w}))$.

Single-TPG: We use temperature $\tau_{global} = 1.5$ during training to encourage exploration, mathematically, $y_{m+1,n}^{\tau_{global}} \sim f^{\tau_{global}}(\mathrm{QGen}(\mathbf{x} \mid \mathbf{w}))$. In the backward pass, the weights are updated using $\mathbf{w} = \mathbf{w} + \alpha(r - b)\nabla_{\mathbf{w}} \ln f(y_{m+1,n}^{\tau_{global}} \mid \mathrm{QGen}(\mathbf{x} \mid \mathbf{w}))$.

Parallel-TPG: For Parallel-TPG, we use two temperatures $\tau_1 = 1.0$ and $\tau_2 = 1.5$ to encourage the exploration. The words are sampled in the forward pass using $y_{m+1,n}^{\tau_1}$, $y_{m+1,n}^{\tau_2} \sim f^{\tau_1,\tau_2}(\mathrm{QGen}(\mathbf{x} \mid \mathbf{w}))$. In the backward pass, the weights are updated using $\mathbf{w} = \mathbf{w} + \Sigma_{k=1}^{2} \alpha(r - b)\nabla_{\mathbf{w}} \ln f(y_{m+1,n}^{\tau_k} \mid \mathrm{QGen}(\mathbf{x} \mid \mathbf{w}))$.

Dynamic-TPG: The last method we evaluated is Dynamic-TPG. We use a heuristic function to calculate the temperature for each word at each time step: $\tau_{m+1,n}^{h} = \tau_{min} + (\tau_{max} - \tau_{min}) \frac{\textit{tf-idf}(y_{m+1,n}^{*}) - \textit{tf-idf}_{min}}{\textit{tf-idf}_{max} - \textit{tf-idf}_{min}}$, where we set $\tau_{min} = 0.5$, $\tau_{max} = 1.5$, and set $\textit{tf-idf}_{min} = 0$, $\textit{tf-idf}_{max} = 8$. After the calculation of $\tau_{m+1,n}^{h}$, we substitute the value into the formula at each time step and sample the next word using $y_{m+1,n}^{\tau_{m+1,n}^{h}} \sim f^{\tau_{m+1,n}^{h}}(\mathrm{QGen}(\mathbf{x} \mid \mathbf{w}))$. In the backward pass, the weights are updated using $\mathbf{w} = \mathbf{w} + \alpha(r - b)\nabla_{\mathbf{w}} \ln f(y_{m+1,n}^{\tau_{m+1,n}^{h}} \mid \mathrm{QGen}(\mathbf{x} \mid \mathbf{w}))$. For all four methods, we use greedy search in evaluation.

5. EXPERIMENT

We first train all the networks in a supervised fashion, and then optimize the QGen and the Guesser model using reinforcement learning. Our implementation [1] uses Torch [29].

5.1. Pretraining

We train all three models using 0.5 dropout [30] during training, using the ADAM optimizer [31]. We use a learning rate of 0.0001 for the Oracle model and the Guesser model, and a learning rate of 0.001 for QGen. All the models are trained with at most 30 epochs and early stopped within five epochs without improvement on the validation set. We report the performance on the test set which consists of images not used in training. We report the game success rate as the performance metric for all three models, which equals to the number of succeeded games divided by the total number of all games. Compared to previous works [6, 8, 32], after supervised training, our models obtain a game success rate of 48.77%, that is 4% higher than state-of-the-art methods [32], which has 44.6% accuracy.

[1] https://github.com/ruizhaogit/GuessWhat-TemperedPolicyGradient

#	Method	Accuracy
1	Strub et al., 2017 [8]	52.30%
2	Strub and de Vries, 2017 [32]	60.30%
3	Our Torch reimplementation of (#2)	**62.61%**
4	(#3) + new QGen (Seq2Seq)	63.47%
5	(#4) + new Guesser (Memory Nets)	68.32%
6	(#5) + new Guesser (REINFORCE)	**69.66%**
7	(#6) + Single-TPG	69.76%
8	(#6) + Parallel-TPG	73.86%
9	(#6) + Dynamic-TPG	**74.31%**

Table 1: Performance comparison and ablation tests

5.2. Reinforcement Learning

We first initialize all models with pre-trained parameters from supervised learning and then post-train the QGen using either REINFORCE or TPG for 80 epochs. We update the parameters using stochastic gradient descent (SGD) with a learning rate of 0.001 and a batch size of 64. In each epoch, we sample each image in the training set once and randomly pick one of the objects as a target. We track the running average of the game success rate and use it directly as the baseline, b, in REINFORCE. We limit the maximum number of questions to 8 and the maximum number of words to 12. Simultaneously, we train the Guesser model using REINFORCE without baseline and using SGD with a learning rate of 0.0001. The performance comparison is shown in Tab. 1.

Ablation Study: From Tab. 1 (#2 & 3), we see that our reimplementation using Torch [29] achieves a comparable performance compared to the original TensorFlow implementation [32]. We use our reimplementation as the baseline.

Upon the baseline, the new QGen model with Seq2Seq structure improves the performance by about 1%, see Tab. 1 (#3 & 4). With the Seq2Seq structure, our QGen model is trained in *question level*. This means that the model first learns to query meaningfully, step by step. Eventually, it learns to conduct a meaningful dialog. Compared to directly learning to manage a strategic conversation, this bottom-up training procedure helps the model absorb knowledge, because it breaks large tasks down into smaller, more manageable pieces. This makes the learning for QGen much easier.

The next improvement is because of our new Guesser model, which uses Memory Network with two-hops attention [17]. The memory and attention mechanisms bring an improvement of 4.85%, as shown in Tab. 1 (#4 & 5). Furthermore, we train the new Guesser model additionally via REINFORCE (#6). In this way, the Guesser and the QGen learn to cooperate with each other and improve the performance by another 1.34%, as shown in Tab. 1 (#5 & 6).

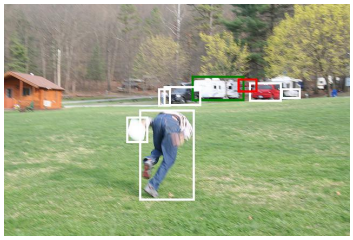
Image	Policy Gradient		Tempered Policy Gradient	
	Is it in left?	No	Is it a person?	No
	Is it in front?	No	Is it a **vehicle**?	Yes
	Is it in right?	Yes	Is it a **truck**?	Yes
	Is it in middle?	Yes	Is it in front of photo?	No
	Is it person?	No	In the left half?	No
	Is it ball?	No	In the middle of photo?	Yes
	Is it bat?	No	Is it to the right photo?	Yes
	Is it **car**?	Yes	Is it in the middle of photo?	Yes
	Status:	Failure	Status:	Success
	Is it in **left**?	No	Is it a giraffe?	Yes
	Is it in front?	Yes	In front of photo?	Yes
	Is it in right?	No	In the **left half**?	Yes
	Is it in middle?	Yes	Is it in the middle of photo?	Yes
	Is it person?	No	Is it **to the left** of photo?	Yes
	Is it giraffe?	Yes	Is it to the right photo?	No
	Is in middle?	Yes	**In the left** in photo?	No
	Is in middle?	Yes	In the middle of photo?	Yes
	Status:	Failure	Status:	Success

Table 2: Some samples generated by our improved models using REINFORCE (left column: "Policy Gradient") and Dynamic-TPG (right column: "Tempered Policy Gradient"). The green bounding boxes highlight the target objects; the red boxes highlight the wrong guesses.

Here, we take a closer look at the improvement brought by TPGs. From Tab. 1, we see that compared to the REINFORCE-trained models (#6), Single-TPG (#7) with $\tau_{global} = 1.5$ achieves a comparable performance. With two different temperatures $\tau_1 = 1.0$ and $\tau_2 = 1.5$, Parallel-TPG (#8) achieves an improvement of approximately 4%. Parallel-TPG requires more computational resources. Compared to Parallel-TPG, Dynamic-TPG only uses the same computational power as REINFORCE does and still gives a larger improvement by using a dynamic temperature, $\tau_t^h \in [0.5, 1.5]$. After comparison, we can see that the best model is Dynamic-TPG (#9), which gives a 4.65% improvement upon new models (#6).

TPG Dialogue Samples: The generated dialogue samples in Tab. 2 can give some interesting insights. First of all, the sentences generated from TPG-trained models are on average longer and use slightly more complex structures, such as "Is it in the middle of photo?" instead of a simple form "Is it in middle?". Secondly, TPGs enable the models to explore better and comprehend more words. For example, in the first task (upper half of Tab. 2), both models ask about the category. The REINFORCE-trained model can only ask with the single word "car" to query about the vehicle category. In contrast, the TPG-trained model can first ask a *more general* category with the word "vehicle" and follows up querying with a *more specific* category "trucks". These two words "vehicle" and "trucks" give much more information than the single word "car", and help the Guesser model identify the truck among many cars. Lastly, similar to the category case, the models trained with TPG can first ask a *larger* spatial range of the object and follow up with a *smaller* range. In the second task (lower half of Tab. 2), we see that the TPG-trained model first asks "In the left half?", which refers to all the three giraffes in the left half, and the answer is "Yes". Then it asks "Is it to the left of photo?", which refers to the second left giraffe, and the answer is "Yes". Eventually the QGen asks "In the left in photo?", which refers to the most left giraffe, and the answer is "No". These specific questions about locations are not learned using REINFORCE. The REINFORCE-trained model can only ask a similar question with the word "left". In this task, there are many giraffes in the left part of the image. The top-down spatial questions help the Guesser model find the correct giraffe. To summarize, the TPG-trained models use longer and more informative sentences than the REINFORCE-trained models.

6. CONCLUSION

Our paper makes two contributions. Firstly, by extending existing models with Seq2Seq and Memory Networks we could improve the performance of a goal-oriented dialogue system by 7%. Secondly, we introduced TPG, a novel class of temperature-based policy gradient approaches. TPGs boosted the performance of the goal-oriented dialogue systems by another 4.7%. Among the three TPGs, Dynamic-TPG gave the best performance, which helped the agent comprehend more words, and produce more meaningful questions. TPG is a generic strategy to encourage word exploration on top of policy gradients and can be applied to any dialog agents.

7. REFERENCES

[1] Rui Zhao and Volker Tresp, "Improving goal-oriented visual dialog agents via advanced recurrent nets with tempered policy gradient," *arXiv preprint arXiv:1807.00737*, 2018.

[2] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press, 1998.

[3] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al., "Human-level control through deep reinforcement learning," *Nature*, 2015.

[5] Antoine Bordes and Jason Weston, "Learning end-to-end goal-oriented dialog," in *International Conference on Learning Representations (ICLR)*, 2017.

[6] Harm de Vries, Florian Strub, Sarath Chandar, Olivier Pietquin, Hugo Larochelle, and Aaron C. Courville, "Guesswhat?! visual object discovery through multi-modal dialogue," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[7] Abhishek Das, Satwik Kottur, José M.F. Moura, Stefan Lee, and Dhruv Batra, "Learning cooperative visual dialog agents with deep reinforcement learning," in *International Conference on Computer Vision (ICCV)*, 2017.

[8] Florian Strub, Harm de Vries, Jérémie Mary, Bilal Piot, Aaron C. Courville, and Olivier Pietquin, "End-to-end optimization of goal-driven and visually grounded dialogue systems," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[9] Zachary Lipton, Xiujun Li, Jianfeng Gao, Lihong Li, Faisal Ahmed, and Li Deng, "Bbq-networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems," *arXiv preprint arXiv:1711.05715*, 2017.

[10] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky, "Deep reinforcement learning for dialogue generation," *arXiv preprint arXiv:1606.01541*, 2016.

[11] Jason D Williams and Geoffrey Zweig, "End-to-end lstm-based dialog control optimized with supervised and reinforcement learning," *arXiv preprint arXiv:1606.01269*, 2016.

[12] Mike Lewis, Denis Yarats, Yann N Dauphin, Devi Parikh, and Dhruv Batra, "Deal or no deal? end-to-end learning for negotiation dialogues," *arXiv preprint arXiv:1706.05125*, 2017.

[13] Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng, "End-to-end reinforcement learning of dialogue agents for information access," *arXiv preprint arXiv:1609.00777*, 2016.

[14] Rui Zhao and Volker Tresp, "Efficient dialog policy learning via positive memory retention," in *IEEE Spoken Language Technology (SLT) (forthcoming)*, 2018.

[15] Ronald J Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, 1992.

[16] Ilya Sutskever, Oriol Vinyals, and Quoc V Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[17] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al., "End-to-end memory networks," in *Advances in neural information processing systems*, 2015, pp. 2440–2448.

[18] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[19] David Carmel and Shaul Markovitch, "Exploration strategies for model-based learning in multi-agent systems: Exploration strategies," *Autonomous Agents and Multi-agent systems*, vol. 2, no. 2, pp. 141–172, 1999.

[20] Ofir Nachum, Mohammad Norouzi, and Dale Schuurmans, "Improving policy gradient by exploring under-appreciated rewards," *arXiv preprint arXiv:1611.09321*, 2016.

[21] Yang Liu, Prajit Ramachandran, Qiang Liu, and Jian Peng, "Stein variational policy gradient," *arXiv preprint arXiv:1704.02399*, 2017.

[22] Sebastian B Thrun, "Efficient exploration in reinforcement learning," 1992.

[23] Andrej Karpathy and Li Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3128–3137.

[24] Christian P Robert, *Monte carlo methods*, Wiley Online Library, 2004.

[25] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman, *Mining of massive datasets*, Cambridge university press, 2014.

[26] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[27] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[28] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[29] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

[30] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[31] Diederik Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[32] Florian Strub and Harm de Vries, "Guesswhat?! models," `https://github.com/GuessWhatGame/guesswhat/`, 2017.