Neural Machine Translation
with Gumbel-Greedy Decoding

Jiatao Gu,[†] **Daniel Jiwoong Im,**[◇] **Victor O.K. Li**[†]

[†]The University of Hong Kong
[†]{jiataogu, wangyong, vli}@eee.hku.hk
[◇]AIFounded Inc.
[◇]daniel.im@aifounded.com

Abstract

Previous neural machine translation models used some heuristic search algorithms (e.g., beam search) in order to avoid solving the maximum a posteriori problem over translation sentences at test phase. In this paper, we propose the *Gumbel-Greedy Decoding* which trains a generative network to predict translation under a trained model. We solve such a problem using the Gumbel-Softmax reparameterization, which makes our generative network differentiable and trainable through standard stochastic gradient methods. We empirically demonstrate that our proposed model is effective for generating sequences of discrete words.

Introduction

Neural machine translation (NMT) (Cho et al. 2014; Sutskever, Vinyals, and Le 2014; Bahdanau, Cho, and Bengio 2014), as a new territory of machine translation research, has recently become a method of choice, and is empirically shown to be superior over traditional translation systems.

The basic scenario of modeling neural machine translation is to model the conditional probability of the translation, in which we often train the model that either maximizes the log-likelihood for the ground-truth translation (teacher forcing) or translations with highest rewards (REINFORCE). Despite these advances, a key problem that still remains with such sequential modeling approaches: once the model is trained, the most probable output which maximizes the log-likelihood during training cannot be properly found at test time. This is because, it involves solving the maximum-a-posteriori (MAP) problem over all possible output sequences. To avoid this problem, heuristic search algorithms (e.g., greedy decoding, beam search) are used to approximate the optimal translation.

In this paper, we address this issue by employing a discriminator-generator framework – we train the discriminator and the generator at training time, but emit translations with the generator at test time. Instead of relying on a non-optimal searching algorithm at test time, like greedy search, we propose to train the generator to predict the search directly. Such a way would typically suffer from non-differentiablity of generating discrete words. Here, we address this problem by turning the discrete output node into a differentiable

node using the Gumbel-Softmax reparameterization (Jang, Gu, and Poole 2016). Throughout the paper, we named this new process of generating sequence of words as *the Gumbel Greedy-Decoding* (GGD). We extensively evaluate the proposed GGD on a large parallel corpora with different variants of generators and discriminators. The empirical results demonstrate that GGD improves translation quality.

Neural Machine Translation

Neural Machine Translation (NMT) models commonly share the auto-regressive property as it is the natural way to model sequential data. More formally, we can define the distribution over the translation sentence $Y = [y^1, ..., y^T]$ given a source sentence $X = [x^1, ..., x^{Ts}]$ as a conditional language model:

$$p(Y|X) = \prod_{t=1}^{T} p(y^t|y^{<t}, X). \tag{1}$$

The conditional probability is composed of an encoder $e_t(\cdot)$ and a decoder network $d_t(\cdot)$ with a softmax layer on top. For notation, we denote the vocabulary size of the target language as K and each word y^t is assigned to an index $k \in [1, K]$. In this paper, we use the one-hot representation for each word, that is, $y_i^t = \mathbb{I}[i = k], i = 1, ...K$. Thus the probability is computed using softmax:

$$p(y^t|y^{<t}, X) = \text{softmax} \left[a \left(z^t; \theta_a \right) \right]^\top \cdot y^t \tag{2}$$

where $\text{softmax}(a)_i = \frac{\exp(a_i)}{\sum_{j=1}^{K} \exp(a_j)}$, and

$$z^t = f(z^{t-1}, y^{t-1}, e^t(X; \theta_e); \theta_d) \tag{3}$$

z_t is the hidden state of the decoder at step t, and a is the energy function which maps the hidden state into a distribution over the vocabulary. The output of the encoder $e^t(X)$ is a time-dependent feature of the source sentence X. Typically, both the encoder and decoder consist of deep recurrent neural networks (with the *soft* attention mechanism integrated) (Bahdanau, Cho, and Bengio 2014; Luong, Pham, and Manning 2015). We use $\theta = \{\theta_a, \theta_d, \theta_e\}$ to denote the parameters of the NMT model.

Training phase

There are two common ways to train NMT models, which are teacher forcing (Williams and Zipser 1989; Sutskever,

Vinyals, and Le 2014; Bahdanau, Cho, and Bengio 2014) and REINFORCE (Williams 1992; Ranzato et al. 2015; Shen et al. 2015; Bahdanau et al. 2016) algorithms [1].

In teacher forcing, the model is trained to maximize the conditional log-likelihood (MLE) of the ground-truth translation Y^* given the source sentence X. In contrast, the REINFORCE algorithm does not rely on the ground-truth translation, but it maximizes the expectation of a global reward function R. In an unified view, the gradients w.r.t the parameters θ for both methods can be seen as:

$$\mathbb{E}_{Y \sim \mathcal{M}} \left[\frac{\partial}{\partial \theta} \log p_\theta(Y|X) \cdot R(Y) \right] \quad (4)$$

where for teacher forcing, $\mathcal{M}$ is the empirical distribution on $Y|X$ and $R(Y) \equiv 1$, while for REINFORCE, $\mathcal{M}$ is p_θ itself and R is used to re-weight the gradients. The primary difference between teacher forcing and REINFORCE is that teacher forcing corrects the translation word-by-word based on the ground-truth prefix, whereas REINFORCE rewards the translated sentence as a whole. The training of teacher forcing is stable but it suffers from the local normalization property (Ranzato et al. 2015). Whereas, although REINFORCE does not have such a problem, it is known to be difficult to train due to the high variance in its gradients.

Test phase

At the test phase, our goal is to make the best translation of the source sentence possible. This process is also known as the *decoding* process. Ideally, we can use Maximum-a-Posteriori (MAP) to find a translation Y which maximizes $\log p_\theta(Y|X)$. Unfortunately, exact MAP inference is intractable due to the exponential complexity in searching. Therefore, we approximate the MAP inference based on some heuristic search-based methods in practice:

Sampling & Greedy Decoding As the model is learned, we can directly perform sampling from the conditional distribution word-by-word, in which case the translation is stochastic. In contrast, rather than maximizing the log-likelihood for the entire translation, greedy decoding simply picks the most likely word at each time step t, resulting in a deterministic translation. However, it is inadequate in practice due to lack of future information.

Beam Search Beam search usually finds better translation by storing S hypotheses with the highest scores ($\prod_{t'=1}^{t} p(y^t|y^{<t}, X)$). When all the hypotheses terminate, it returns the hypothesis with the highest log-probability. Despite its superior performance compared to greedy decoding, the computational complexity grows linearly w.r.t. $|S|$, rendering it less preferable in production environment.

Discriminator-Generator framework

The major discrepancy between training and testing time is that we cannot leverage the full power of our trained NMT

model during testing. Here, we propose to train a separate generative network that will reduce the potential mismatch between the training and testing phases. Let us first portray the training and test procedure in terms of the discriminator-generator framework as following:

- *NMT-discriminator* - measures the log-likelihood at word level - $\log p_\theta(Y|X)$ - given the source sentence X and a translation Y.
- *NMT-generator* - generates the translation by taking the output of the word as an input to next step recursively - $Y = G_\phi(X)$ - given the source sentence X. (G is usually a search-based method).

We train the generative network through a "GAN-like" discriminator-generator-framework, where the output of the NMT-generator[2] gets fed to the NMT-discriminator (see Fig. 1 (b)). We propose to learn the parameters of generator ϕ by maximizing the NMT discriminator's score,

$$J(\phi) = \mathbb{E}_{Y \sim G_\phi} \log p_\theta(Y|X) \quad (5)$$

and the gradient w.r.t ϕ is computed using chain rule,

$$\left. \frac{\partial \log p_\theta(Y|X)}{\partial Y} \right|_{Y = G_\phi(X)} \cdot \frac{\partial G_\phi(X)}{\partial \phi} \quad (6)$$

In practice, we can set the initial parameters of the generator to be the same as the discriminator's parameters. Note that the discriminator and the generator share the same parameters, i.e., $\theta = \phi$, and the generator is never trained in the traditional NMT framework.

Unfortunately, optimizing the generator with Eq. 5 and 6 involves operations, such as sampling or argmax, that are non-differentiable w.r.t (discrete) words. Therefore, we cannot leverage the backpropagation algorithm (Rumelhart, Hinton, and Williams 1986). Here, we solve this problem by incorporating the Gumbel-Softmax relaxation into it(Section).

Gumbel-Greedy Decoding

In this section, we show how to train the generator w.r.t the discriminator's output using the idea of *Gumbel-Greedy Decoding*, where we apply the Gumbel-Softmax based reparameterization trick in sampling of the NMT-generator. The main idea is to turn the stochastic node (the last layer of the generator network) into a differentiable function of the network parameters with an independent random variable.

Sampling as Gumbel-Greedy Decoding

The Gumbel-Max Trick (Gumbel 1954) transforms sampling from a categorical distribution to an optimization problem, and visa versa. That is to say, $y \sim p_\theta(y|y^{<t}, X) = \text{softmax}(a)$ in Eq. 2 is equivalent to [3]:

$$y = \text{argmax}(g + a), g \sim \text{Gumbel i.i.d.} \quad (7)$$

where $\text{argmax}(x)_i = \mathbb{I}[x_i = \max(x)]$, and each element in g can be computed using the inverse transform sampling of an auxiliary random uniform variable $u_i \sim \mathcal{U}(0, 1)$,

---

[1]For simplicity, previous efforts using reinforcement learning to train NMT are treated as variants of REINFORCE.

[2]The term generator and decoder are used interchangeably.

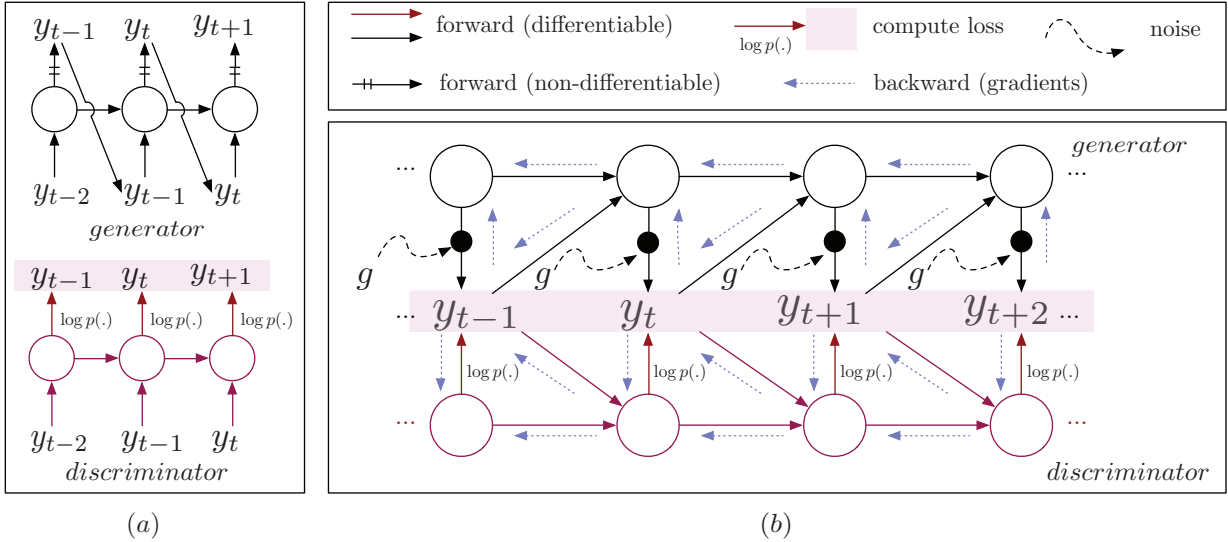[3]We omit the time-step mark t for simplicity.

Figure 1: (a) An example illustrating the two functions of a NMT model: *discriminator* and *generator*, (b) An illustration of computational flow for the training of *Gumbel-Greedy Decoding*. The cost function uses the same architecture of the generator to compute the log-likelihood.

$g_i = -\log(-\log(u_i))$. Since the Gumbel noise g and sampled words are independent, we can simply break down the sampling into a two-step process:

- Sample a noise g_t from Gumbel distribution at each step;

- Perform the greedy decoding based on a noise-biased distribution in Eq. 7.

Note that the Gumbel-max trick does not change the non-differentiability of sampling,

Gumbel-Softmax Relaxation (Maddison, Mnih, and Teh 2016; Jang, Gu, and Poole 2016) proposed a reparameterization trick for discrete random variables based on Gumbel-Softmax where

$$\hat{y} = \text{softmax}((g + a)/\tau), g \sim \text{Gumbel i.i.d.} \quad (8)$$

where $\tau \in (0, \infty)$ is the temperature. The softmax function approaches argmax operations as $\tau \to 0$, and it becomes uniform when $\tau \to \infty$. Thus, the samples are no longer one-hot vectors. With the Gumbel-Softmax relaxation, we can easily derive the partial gradient estimator $\partial\hat{y}/\partial a$ of Gumbel-Softmax as:

$$\frac{\partial\hat{y}_i}{\partial a_j} = \hat{y}_i \left(\delta_{ij} - \hat{y}_j\right)/\tau \quad (9)$$

where $\delta_{ij} = \mathbb{I}[i = j]$. This allows us to train the NMT model using the backpropagation algorithm. Note that according to Eq. 9, $\lim_{\tau \to 0} [\partial\hat{y}_i/\partial a_j] = 0$ (or $\pm\infty$ if more than 2 words achieve the maximum energy simultaneously), which makes training with backpropagation impossible for $\tau \to 0$.

Straight-Through (ST) Gumbel Nonetheless, there still remains a challenge to overcome before we can apply Gumbel-Softmax reparameterization to NMT. In language

modeling, the embedding vector is chosen from the look up table based on the generated word, and is emitted in the next time step. However, the Gumbel-Softmax relaxation leads to a mixture of embedding vectors, in turn causing a mixing error. Furthermore, such mixing errors get accumulated over time as the errors are propagated forwards through the recurrent neural network. This causes future word generation to deteriorate even with a small temperature τ, especially when we are using a pre-trained model.

In order to avoid the problem of mixing and propagating word embedding errors over time, we apply *the straight-through* version of the Gumbel-Softmax estimator (Jang, Gu, and Poole 2016), or called ST-Gumbel. During the forward phase, we use the Gumbel-Max in Eq. 7, while computing the gradient of the Gumbel-Softmax in Eq. 9, i.e., $\hat{y}_t$ in Eq. 9 is replaced by y_t in Eq. 7. Obviously, the ST-Gumbel estimator is biased due to the sample mismatch between the forward and backward passes. However, we find that it works empirically.

Learning By putting all together, we can derive the basic learning algorithm for Gumbel-greedy decoding. We estimate the gradient from a differentiable cost function $R(Y)$ w.r.t. ϕ:

$$\mathbb{E}_{Y \sim G_\phi} \left[\frac{\partial}{\partial\phi}R(Y)\right] \approx \mathbb{E}_{g \sim P} \left[\frac{\partial R}{\partial Y}\frac{\partial\hat{Y}}{\partial\phi}\right] \quad (10)$$

where we use P to represent the noise distribution over all time steps. From the equation, it is clear that such approximation holds by assuming $\partial Y/\partial\hat{Y} \approx 1$ which however is not always true. In practice, we can use any differentiable cost function that evaluates the goodness of the generation output. For instance, a fixed language model, a critic that

predicts BLEU scores, or the NMT-discriminator (Eq. 5). We will discuss this in later sections.

Arbitrary Decoding Algorithms as Gumbel-Greedy Decoding

Inference on Gumbel The Gumbel-Max trick indicates a general formulation that can present any decoding algorithms as Gumbel-Greedy Decoding:

$$y = \text{argmax}\,(g + a)\,, g \sim Q \tag{11}$$

where Q represents a special distribution that generates this word, which is typically unknown. Note that when we choose $Q = P$, the decoding algorithm degenerates into sampling. However, as discussed in (Maddison, Tarlow, and Minka 2014), given the trajectory of decoded words, we can efficiently infer its corresponding Gumbel noise g using a top-down construction algorithm as $g = g^* - a$, and:

$$g_i^* = \begin{cases} g', & y_i \text{ is selected} \\ g' - \log\left[1 + e^{g' - \tilde{g}_i}\right], & \text{otherwise} \end{cases} \tag{12}$$

where

$$\tilde{g}_i = -\log(-\log(u_i)) + a_i, u_i \sim \mathcal{U}(0,1)$$

and the "top-gumbel"

$$g' = -\log(-\log(u)) + \log\left(\sum_i \exp(a_i)\right), u \sim \mathcal{U}(0,1)$$

Such inference is a special case of the core top-down algorithm of A* sampling.

Learning The above inference algorithm shows that, after running any decoding algorithm $G_\phi(X)$ (e.g. greedy decoding, beam search, etc.), we can always infer corresponding noise g^t at each time step. Although in such cases the inferred noise does depend on the translation, which breaks the requirement of reparameterization trick, the decoding methods we use are usually deterministic methods. That is, $p(g, Y|X) \approx p(g|X)p(Y|X)$. It is possible to train the deterministic generator as an equivalent Gumbel-greedy decoding using Eq. 10.

Gumbel-Greedy Decoding for Discriminator-Generator Framework

We can finally conclude the learning algorithm for the proposed Discriminator-Generator framework using GGD, by simply setting the discriminator's output $\log p_\theta(Y|X)$ as the cost function in Eq. 10, as shown in Fig 1(b), where we illustrate the computational flow of the whole framework. Note that the non-dfferentiable path is replaced by a differentiable path with an additional noise due to GGD, and gradient (though biased) flows can freely go through both directions of the discriminator and the generator, sufficiently communicating all useful information for learning. The overall algorithm for learning with GGD is found in Algorithm 1.

With Regularization One issue we observed in practice is that, directly optimizing the discriminator's output is not stable for learning the generator with GGD. Fortunately, we can stabilize the optimization by adding an entropy term in the cost function w.r.t ϕ:

$$\mathbb{E}_{G_\phi}\left[\log p_\theta(Y|X)\right] - \mathbb{E}_{G_\phi}\left[\log p_{\phi'}(Y|X)\right] \tag{13}$$

where we use ϕ' to represent a copy of the current parameters ϕ and make it as a "discriminator". Note that gradients w.r.t ϕ will not flow into ϕ'.

Adversarial Learning Even though it is possible to learn the generator with a fixed discriminator, the proposed framework also allows to optimize both the discriminator and the generator in an adversarial way (Goodfellow et al. 2014). In particular, we take a similar formulation of the energy-based generative adversarial nets (Zhao, Mathieu, and LeCun 2016) where in our case we use the discriminator's output as the energy to distinguish the ground-truth translation and the generator's generation, w.r.t θ:

$$\mathbb{E}_D\left[\log p_\theta(Y|X)\right] - \mathbb{E}_G\left[\log p_\theta(Y|X)\right] \tag{14}$$

where D is the empirical distribution of real translation. In practice, we alternate the training of the generator and the discriminator iteratively.

Algorithm 1 Gumbel-Greedy Decoding

Require: discriminator p_θ, generator G_ϕ, $N_d \geq 0$, $N_g > 0$
1:   Train θ using MLE/REINFORCE on training set D;
2:   Initialize ϕ using θ;
3:   Shuffle D twice into D_θ and D_ϕ
4:   **while** stopping criterion is not met **do**
5:      **for** $t = 1 : N_g$ **do**   // *learn the generator*
6:         Draw a translation pair: $(X, \_) \sim D_\phi$;
7:         Obtain $Y, \hat{Y} = \text{GUMBELDEC}(G, X)$
8:         Compute forward pass $\sim X, Y$ with Eq. 13
9:         Compute backward pass $\sim X, \hat{Y}$, update ϕ
10:     **for** $t = 1 : N_d$ **do**   // *learn the discriminator*
11:        Draw a translation pair: $(X, Y^*) \sim D_\theta$;
12:        Obtain $Y, \_ = \text{GUMBELDEC}(G, X)$
13:        Compute forward pass $\sim X, Y, Y^*$ with Eq. 14
14:        Compute backward pass $\sim X, Y, Y^*$, update θ

Function: $\text{GUMBELDEC}(G, X)$
1:   **if** $G = $ 'sampling' **then**
2:      Sample $g \sim$ Gumbel i.i.d.
3:      Obtain $Y, \hat{Y}$ with Eq. 7 and Eq. 8
4:   **else**
5:      Obtain $Y = G(X)$
6:      Infer g with Eq. 12
7:      Obtain $\hat{Y}$ with Eq. 8
8:   Return $Y, \hat{Y}$

Experiments

Experimental Setup

Dataset We consider translating – Czech-English (Cs-En) and German-English (De-En) – language pairs for both directions with a standard attention-based neural machine translation system (Bahdanau, Cho, and Bengio 2014). We use the

parallel corpora available from WMT'15[4] as a training set. We use newstest-2013 for the validation set to select the best model according to the BLEU scores and use newstest-2015 for the test set. All the datasets were tokenized and segmented into sub-word symbols using byte-pair encoding (BPE) (Sennrich, Haddow, and Birch 2015). We use sentences of length up to 50 subword symbols for teacher forcing and 80 symbols for REINFORCE, GGD and testing.

Architecture We implement the NMT model as an attention-based neural machine translation model whose encoder and decoder recurrent networks have 1,028 gated recurrent units (GRU, Cho et al., 2014) each. For the encoder, a bidirectional RNN is used and we concatenate the hidden states from both directions to build the context at each step. For the decoder, a single layer feed-forward neural network (512 hidden units) is used to compute the attention scores. Both source and target symbols are projected into 512-dimensional embedding vectors. The same architecture is shared by the NMT-discriminator and the NMT-generator.

Baselines We set our baseline to be NMT model trained with teacher forcing and REINFORCE algorithm. Our NMT model was trained with teacher forcing method (Maximum Likelihood) for approximately 300,000 updates for each language pairs. These networks were trained using Adadelta (Zeiler 2012). We further fine-tuned these models using REINFORCE with a smoothed sentence-level BLEU (Lin and Och 2004) as reward following similar procedures in (Ranzato et al. 2015). We denote the former trained model as θ_{ML} and the additionally trained model using REINFORCE as θ_{RL}.

Additionally, we explored the Straight-Through (ST) estimator (Bengio, Léonard, and Courville 2013; Chung, Ahn, and Bengio 2016) and compared with the ST-Gumbel that GGD uses for passing the gradients. The difference between the two is that, we use the output distribution softmax (a/τ) instead of softmax$((g+a)/\tau)$ in Eq. 8 in the original ST estimator. The ST estimator, as a special case of ST Gumbel estimator, is independent of the choice of the selected word in the forward pass.

Pre-training In our experiments, we use pre-trained models from the baseline θ_{ML} for the discriminative networks for training generative (decoding) network. It is possible to start a generator ϕ from scratch for generating translation, and yet it has been shown to be effective if the generator is continually learned from the initialization of a pre-trained discriminator (Ranzato et al. 2015; Shen et al. 2015; Bahdanau et al. 2016; Lamb et al. 2016). Because our learning algorithm requires sampling from the generator, the searching space is extensive for a randomly initialized generator to output any meaningful translation to learn from. In our experiments, we observed that initializing the parameter of the generator $\phi = \theta_{ML}$ worked consistently better whether we choose a stochastic generator for sampling or a deterministic one for greedy decoding.
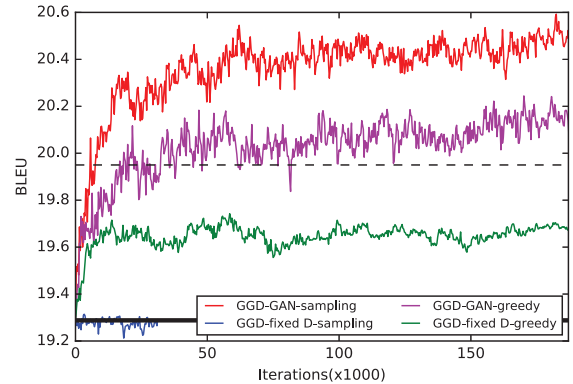
---

[4]http://www.statmt.org/wmt15/

Figure 2: Comparison of greedy BLEU scores on the validation set of Cz-En, achieved by two generators that are learned to optimize a discriminator initially trained with teacher forcing. "GAN" refers to the discriminator being iteratively trained together with the generator, while "fixed D" refers to the discriminator being fixed. The straight black line and the black dashed lines are the BLEU scores achieved by the baseline models learned with teacher forcing and REINFORCE.

Learning of GGD We report the results of generator trained with sampling and greedy decoding, respectively based on Eq. 13. We find that learning using RMSProp (Tieleman and Hinton 2012) is most effective with the initial learning rates of 1×10^{-5}. It is also possible to continually learn the discriminator according to Eq. 14. The generator usually gets updated much more than the discriminator. In our experiments, we used 10 updates for the generator for every discriminator's update.

We denote GGD-GAN for the model where the discriminator and the generator is jointly trained. We denote GGD-Fixed-D for the model where only the generator is trained with fixed discriminator.

Results and Analysis

In our first experiment, we examine whether the GGD-GAN is more effective compare to GGD-fixed D. Fig. 2 presents the results of training based on both sampling and greedy methods. We observe that both GGD-GAN and GGD-Fixed-D give much higher than the two baseline models, θ_{ML} and θ_{RL}, by $\approx +1.3$ and $\approx +0.6$ respectively. Furthermore, the curves in Fig 2 shows that we get the best validation BLEU score when the discriminator is trained together with a stochastic generator with a adversarial loss. The reason why GAN style of training works better than the fixed discriminator training is because, we cannot get any additional information that helps the generator when we are just training the generator (see the blue curve in Fig. 2) when we start with the same generator and the discriminator.

Importantly, we notice that the generator with GGD always improves the score compared to the original model. This illustrates that even when the trained discriminator is not optimal, the discriminator can be jointly trained with

(a) comparison with variant temperature

(b) comparison w/o entropy regularization

(c) comparison of greedy decoding
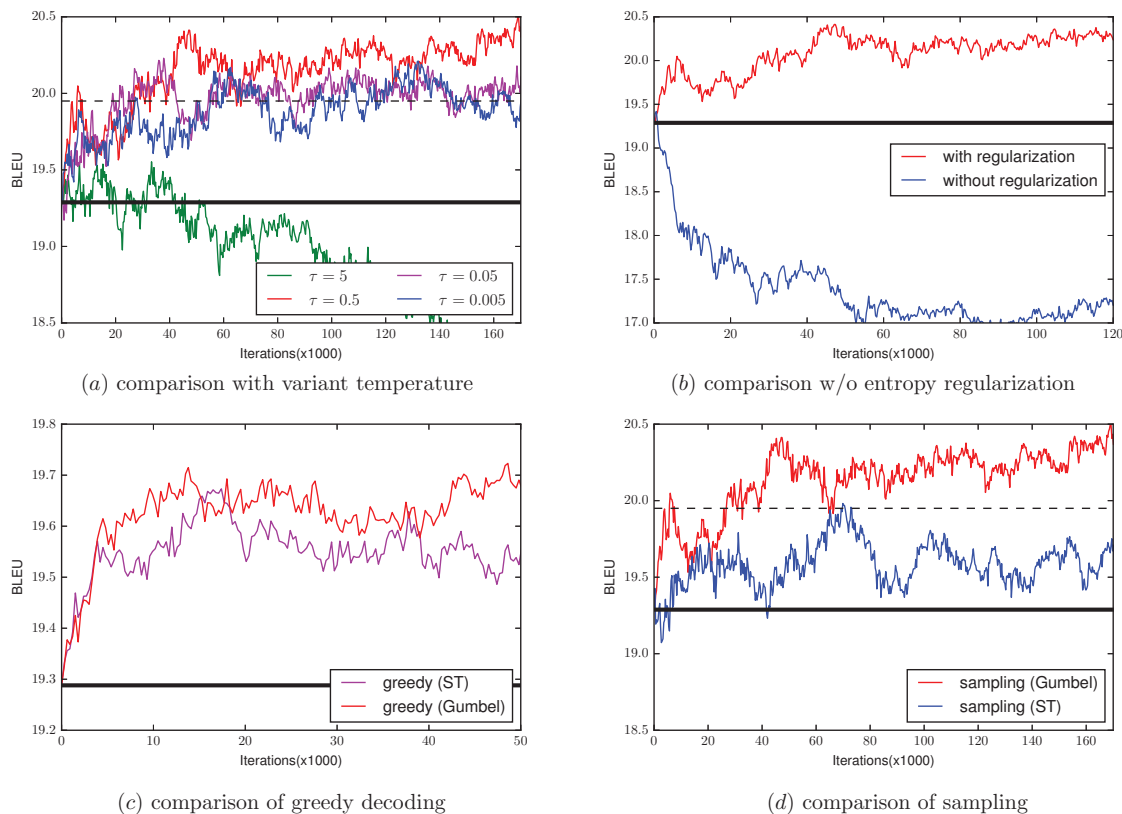
(d) comparison of sampling

Figure 3: Comparison of greedy BLEU scores on the validation set of Cs-En. Both (a) and (b) are achieved by stochastic generators that are learned to optimize a discriminator trained with REINFORCE. (c) shows the comparison of learning the generator of greedy decoding w.r.t. the teacher-forcing discriminator; (d) shows the comparison of learning the generator of sampling w.r.t. the REINFORCE discriminator. For all sub-figures, the black straight line and the black dashed line mean the BLEU scores achieved by the baseline models learned with teacher forcing and REINFORCE, respectively.

the generator again to achieve better score. In fact, just by training greedy decoding on generator enhances the BLEU score as shown in Fig. 3 green curve. Finally, we get the most improvement when we use GGD with sampling instead of GGD with greedy decoding.

Importance of Regularization We experimentally demonstrate the effectiveness of entropy regularization from Section . Fig. 3 (b) presents the performance with and without the entropy regularization term. This figure illustrates that the generator drops dramatically and only optimize the discriminator when the entropy term is removed. We hypothesize that one reason could be that the output distribution of a pretrained generator became highly peaked, and therefore, it is sensitive to small changes. Thus, just relying on a discriminator network, which act as a positive force that *pushes* the distribution go to a better direction is not sufficient. Rather, we need the regularizer that act as a negative force, which distributes the probability mass, is necessary. Lastly, we also note that Eq. 13 can also be seen as the minimizing the Kullback-Leibler (KL) divergence between p_ϕ and $p_{\theta_{RL}}$ and it achieves the optimal as $p_\phi = p_{\theta_{RL}}$.

The sensitivity analysis w.r.t the temperature τ One of the extra hyperparameter that is added from GGD is the Gumbel-Softmax temperature τ rate. Here, we explore how the changes in the temperature effect the performance. The four different temperature rates $\{5, 0.5, 0.05, 0.005\}$ were used in the experiment.

The curves in Fig. 3 (a) demonstrate that the best result is achieved when $\tau = 0.5$. A smaller τ leads to a vanishing gradient problem. In contrast, a larger τ also leads to unstable training. We see that the performance curve drops dramatically at $\tau = 5$. We speculate that this is due to the bias in the estimator. As the bias inside the estimator depends on the amount of forward-backward mismatch $\Delta y = y - \hat{y}$, which is proportional to the temperature we use. Jang, Gu, and Poole (2016) suggests to anneal the temperature over the training time. However, we did not find the annealing technique help in practice at least for NMT. All of our models were trained with temperature rate of 0.5 in the other experiments.

ST versus ST-Gumbel Last but not least, we compared the original Straight-Through (ST) estimator (Bengio, Léonard,

	SRC	Izraelské děti se po válce v Gaze vrací do školy
	REF	Israeli Children Return to School After Gaza War
	MLE	Israeli children , after the war in Gaza , go back to school
	GGD	Israeli children return to school after the Gaza war
	SRC	Podle mých informací se takové pušky vyráběly až ke konci války .
	REF	When we got close to the clay bottom , we reached a barrel , " Radomil Novák recalled .
	MLE	We came to the barrel as we approached the clay day , ' said Radomil Novák .
	GGD	We were on the gun when we got close to the clay day , ' said Radomil Novák .
	SRC	Jde o další ze série podobných zkoušek , které izolovaný severokorejský režim provedl v posledních týdnech .
	REF	This year North Korea carried out an unusually high number of rocket and artillery tests .
	MLE	North Korea , this year , was unusually large in missiles and artillery tests .
	GGD	North Korea , this year , was unusually large amounts of the rocket and artillery tests .

Figure 4: Three Cs-En examples in which the difference between the MLE and the GGD-GAN is large.

	Model	DE-EN	EN-DE	CS-EN	EN-CS
	MLE	21.63	18.97	18.90	14.49
Greedy	RL	22.56	19.32	19.45	15.02
	GGD	**23.27**	**19.81**	**20.62**	**16.04**
	MLE	24.46	21.33	21.20	16.20
Beam	RL	25.12	**22.13**	21.92	17.02
	GGD	**25.32**	21.97	**22.47**	**17.64**

Table 1: The greedy decoding and the beam-search performance of models trained with GGD-GAN against MLE and REINFORCE (referred to RL). BLEU scores are calculated on the test sets.

and Courville 2013; Chung, Ahn, and Bengio 2016) with ST-Gumbel. Since ST is just a special case of ST-Gumbel, we can run all the experiments in the same way and simply drop the Gumbel noise term when computing the backward pass. As shown in Fig. 3 (c) and (d), we have two experiments using these two estimators, training greedy decoding and sampling, respectively.

We observe that ST-Gumbel works better than the original ST estimator in both cases, especially when training the generator with sampling. This is because the backward pass of the ST-estimator is independent of the word that we choose in the forward pass. This is especially problematic for sampling-based compare to greedy-based, because we can get a sampled word that has a relatively small probability in the output distribution. In contrast, the ST-Gumbel always sets the selected word with the highest score in the Gumbel-Softmax (Eq. 8) by adding the noise. Consequently, this reduces the bias compared with the ST-estimator and makes the learning more stable.

Final Results Based on the above experiments, we find the most proper training setting for GGD is when we i) jointly training the discriminator and the generator, and ii) use sampling-based generator with additional entropy regularization. We report the final performance on all four language pairs with these settings in Table 1 Both BLEU scores of greedy decoding and beam-search (size=5) are reported. It is clear that the generators trained with the proposed GGD algorithm can consistently outperforms the baseline models in the most of the cases. Examples of the proposed GGD-GAN that outputs better translation results can be found in Fig. 4.

Related Work

There has been several work on training to solve decoding problem in NLP (Shen et al. 2015; Ranzato et al. 2015; Wiseman and Rush 2016). Recently, there has been a work that came out independently of ours on learning to decode. Li, Monroe, and Jurafsky (2017) proposed to train a neural network that predicts an arbitrary decoding objective given a source sentence and a partial hypothesis or a prefix of translation. They use it as an auxiliary score in beam search. For training such a network, referred to as a Q network in their paper, they generate each training example by either running beam search or using a ground-truth translation (when appropriate) for each source sentence. This approach allows one to use an arbitrary decoding objective, and yet it still relies heavily on the log-probability of the underlying neural translation system in actual decoding.

The proposed framework and the GGD algorithm are also directly motivated by Generative Adversarial Networks (GANs), which are one of the popular generative models that consist of discriminative and generative networks (Goodfellow et al. 2014). Energy-based GAN was later introduced, which uses the energy as the score function rather than binary score (i.e., predicting whether the input is real or fake) (Zhao, Mathieu, and LeCun 2016). The GAN style of training has been widely applied in vision domain (Radford, Metz, and Chintala, 2015; Im et al., 2016a,b). There are only few works where GAN style of training is applied to sequential modeling (Yu et al. 2016; Kusner and Hernández-Lobato 2016) and machine translation. To the best of our knowledge, we are the first to apply Gumbel-softmax relaxation in a generator-discriminator framework for training neural machine translation. The closest work to ours is (Kusner and Hernández-Lobato 2016), which applies GAN for modeling simple sequences, and they also applied the Gumbel-Softmax to GAN. However, their problem setup and the training framework differ from ours in a sense that, i) their discriminator is exactly the same as the classical GAN, whereas our GAN is more close to the energy-based GAN; ii) they only apply to synthetic dataset, whereas we apply it to NMT with a large scale parallel corpora. The application of Gumbel distribution can also be seen in (Papandreou and Yuille 2011).

Conclusion

We studied learning the neural machine translation decoding in a differentiable way. Our solution was to use the Gumbel-Softmax reparameterization trick, which makes our generative network differentiable and can be trained through standard stochastic gradient methods. We empirically demonstrate that our proposed model is effective for generating sequence of discrete words. In the future work, we hope to explore adversarial learning using different reward functions with GGD. This includes both differentiable and non-differentiable rewards.

Acknowledgments

References

Bahdanau, D.; Brakel, P.; Xu, K.; Goyal, A.; Lowe, R.; Pineau, J.; Courville, A.; and Bengio, Y. 2016. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chung, J.; Ahn, S.; and Bengio, Y. 2016. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, 2672–2680.

Gumbel, E. J. 1954. Statistical theory of extreme values and some practical applications: a series of lectures.

Im, D. J.; Kim, C. D.; Jiang, H.; and Memisevic, R. 2016a. Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*.

Im, D. J.; Ma, H.; Kim, C. D.; and Taylor, G. 2016b. Generating adversarial parallelization. *arXiv preprint arXiv:1612.04021*.

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Kusner, M. J., and Hernández-Lobato, J. M. 2016. GANS for Sequences of Discrete Elements with the Gumbel-softmax Distribution. *ArXiv e-prints*.

Lamb, A. M.; GOYAL, A. G. A. P.; Zhang, Y.; Zhang, S.; Courville, A. C.; and Bengio, Y. 2016. Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, 4601–4609.

Li, J.; Monroe, W.; and Jurafsky, D. 2017. Learning to decode for future success. *arXiv preprint arXiv:1701.06549*.

Lin, C.-Y., and Och, F. J. 2004. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, 605. Association for Computational Linguistics.

Luong, M.-T.; Pham, H.; and Manning, C. D. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Maddison, C. J.; Tarlow, D.; and Minka, T. 2014. A* sampling. In *Advances in Neural Information Processing Systems*, 3086–3094.

Papandreou, G., and Yuille, A. L. 2011. Perturb-and-MAP random fields: Using discrete optimization to learn and sample from energy models. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, 193–200. IEEE.

Radford, A.; Metz, L.; and Chintala, S. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Ranzato, M.; Chopra, S.; Auli, M.; and Zaremba, W. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Rumelhart, D.; Hinton, G.; and Williams, R. 1986. Learning representations by back-propagating errors. *Nature* 323–533.

Sennrich, R.; Haddow, B.; and Birch, A. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Shen, S.; Cheng, Y.; He, Z.; He, W.; Wu, H.; Sun, M.; and Liu, Y. 2015. Minimum risk training for neural machine translation. *arXiv preprint arXiv:1512.02433*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Tieleman, T., and Hinton, G. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4(2).

Williams, R. J., and Zipser, D. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1(2):270–280.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Wiseman, S., and Rush, A. M. 2016. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*.

Yu, L.; Zhang, W.; Wang, J.; and Yu, Y. 2016. Seqgan: sequence generative adversarial nets with policy gradient. *arXiv preprint arXiv:1609.05473*.

Zeiler, M. D. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhao, J.; Mathieu, M.; and LeCun, Y. 2016. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*.