A Appendix A

A.1 Implementation Details for Review Generators

Recurrent Neural Networks (RNNs) directly model the generation process of text sequences, and provide an end-to-end solution to learning the generating function from large quantities of data. These networks maintain a hidden layer of neurons with recurrent connections to their own previous values, which in theory gives them the potential to model long span dependencies. For an input sequence $x = x_1, x_2, \ldots, x_t$, the hidden state h_t which summarizes the information of the entire sequence up to timestep t is recursively updated as $h_t = f(h_{t-1}, x_t)$, where $f(.,.)$ denotes a non-linear transformation function. The overall probability of the sequence is calculated as:

$$p(x) = \prod_{t=1}^{T} p(x_t|h_{t-1}), \tag{1}$$

and the probability of generating the next word x_{t+1} given its low dimensional continuous representation $O_{x_{t+1}}$ and input sequence x_t is defined as:

$$p(x_{t+1}|x \leq t) = p(x_{t+1}|h_t) \propto \exp(O_{x_{t+1}}^T h_t) \tag{2}$$

However, in practice the gradient computation is difficult to propagate back in time due to exploding or vanishing gradients (Hochreiter et al., 2001), (Bengio et al., 1994), making the learning of arbitrarily long phenomena challenging in RNNs. Long Short Term Memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997) effectively address these limitations by relying on a memory state and gating functions to control the flow of the information throughout the network – and in particular what information is written to the memory state, what information is read from the memory state, and what information is removed (or forgotten) from the memory state. The mathematical formulation of LSTM units can be expressed as follows:

$$\begin{aligned}
i^{(t)} &= \sigma(W^{(i)}x^{(t)} + U^{(i)}h^{(t-1)}) && \text{(Input gate)} \\
f^{(t)} &= \sigma(W^{(f)}x^{(t)} + U^{(f)}h^{(t-1)}) && \text{(Forget gate)} \\
o^{(t)} &= \sigma(W^{(o)}x^{(t)} + U^{(o)}h^{(t-1)}) && \text{(Output gate)} \\
\widetilde{c}^{(t)} &= \tanh(W^{(c)}x^{(t)} + U^{(c)}h^{(t-1)} && \text{(New memory cell)} \\
c^{(t)} &= f^{(t)} \circ \widetilde{c}^{(t-1)} + i^{(t)} \circ \widetilde{c}^{(t)} && \text{(Final memory cell)} \\
h^{(t)} &= o^{(t)} \circ \tanh(c^{(t)})
\end{aligned} \tag{3}$$

In the above set of equations, the input word $x^{(t)}$ and the past hidden state $h^{(t-1)}$ are used to generate new memory $\widetilde{c}^{(t)}$ which includes features of the new word $x^{(t)}$ without prior determination of whether $x^{(t)}$ is important and worth keeping. The role of the input gate is to check whether it is sensible to store the new input word given the word $x^{(t)}$ itself and the past hidden state $h^{(t-1)}$; the input gate produces $i^{(t)}$ as output, which encapsulates the worthiness decision of preserving the input information. Similarly to the input gate, the forget gate also determines the usefulness of a word by inferring whether the past memory cell is used to compute the current memory cell by looking at the input word word $x^{(t)}$ itself and the past hidden state $h^{(t-1)}$; it produces $f^{(t)}$ as output, which encapsulates the worthiness decision of preserving the past memory cell. In the final memory generation stage, the advice of the input gate $i^{(t)}$ to gate the new memory $\widetilde{c}^{(t)}$ and the advice of the forget gate $f^{(t)}$ to forget the past memory $\widetilde{c}^{(t-1)}$ are both considered, and the two results are summed up to produce the final memory $c^{(t)}$. The output gate is used to separate the hidden state h^t from the final memory of the network $c^{(t)}$. Given that every state of the LSTM is relying on hidden states and that the final memory $c^{(t)}$ contains a lot of information not necessarily required to be saved in the hidden state, the output gate discriminatively assesses which parts of the memory $c^{(t)}$ should be kept inside the hidden state h^t. In our experiments we employ an LSTM generative model trained at word level. Sampling from a trained word language model can be done in two ways: beam search (Bahdanau et al., 2014) and random sampling (Graves, 2013). Following (Tang et al., 2016), we use random sampling with different values for the temperature parameter. Sampling from the LSTM model with a high temperature results in the model generating diverse samples at the

cost of introducing some mistakes, while small temperatures generate conservative samples without a lot of content diversity. In our experiments, we empirically set the temperatures to the following values: 1.0, 0.7 and 0.5.

RNNs, and LSTMs in particular, have become the standard for modeling machine learning problems that involve temporal and sequential data including text. The data is modeled via a fully-observed directed graphical model, where the distribution over a discrete time sequence $y_1, y_2, \ldots, y_T$ is decomposed into an ordered product of conditional distributions over tokens:

$$P(y_1, y_2, \ldots, y_T) = P(y_1) \prod_{t=1}^{T} P(y_t | y_1, \ldots, y_{t-1}) \tag{4}$$

For models with recurrent connections from their outputs leading back into the model, *teacher forcing* (Williams and Zipser, 1989) is the most popular training strategy. This procedure emerges from the maximum likelihood criterion, in which at training time $t+1$ the model receives as input the ground truth output y^t:

$$\log p(y^{(1)}, y^{(2)} | x^{(1)}, x^{(2)}) = \log p(y^{(2)} | y^{(1)}, x^{(1)}, x^{(2)}) + \log p(y^{(1)} | x^{(1)}, x^{(2)}) \tag{5}$$

The model in Equation 5 above illustrates the conditional maximum likelihood criterion at timestep $t = 2$. The model is trained to maximize the conditional probability of $y^{(2)}$ given the sequence x generated so far and the previous $y^{(1)}$ value. Therefore, maximum likelihood specifies that at training time the previous token generated by the model is replaced with ground-truth examples y_t that are fed back into the model for predicting outputs at later time steps. Feeding back ground truth samples at training time forces the RNN to stay close to the ground-truth sequence. However, at inference time, the ground truth sequence is no longer available conditioning, and each y_t is generated by the model itself (i.e. sampled from its conditional distribution over the sequence given the previously generated samples). This discrepancy between training time and inference time causes errors in the model predictions that accumulate and amplify quickly over the generated sequence as the model is in a part of the state space it has never seen during training time. Small prediction errors compound in the RNN's conditioning context, and as the generated sample starts to diverge from sequences it has seen during training, the prediction performance of the RNN worsens (Lamb et al., 2016).

To alleviate this problem, Bengio et al (Bengio et al., 2015) propose *Scheduled Sampling (SS)*, a learning strategy for training RNNs which mixes inputs from the ground-truth sequence with inputs generated by the model itself at training time. SS relies on curriculum learning (Bengio et al., 2009) to change the training process from a fully guided scheme using the true previous token to a less guided scheme mostly using the generated token. The choice of replacing the ground truth with the model's prediction is determined by a coin flip with some probability, independently for each token. The probability of using the ground truth is set to a high value initially. As the model gradually keeps improving, samples from the model become more frequent and the model is partially fed with its own synthetic data as prefix in a similar way to inference mode. Therefore, the training objective is slowly changed from an easy task where the previous token is known, to a realistic task where the previous token is provided by the model itself. The scheduled sampling training scheme is meant to make the model more robust and forces it to deal with its own mistakes at training time, in a similar way to inference time. However, as the model generates several consecutive tokens y_t-s, it is not clear whether the correct target distribution remains the same as in the ground truth sequence. The authors propose two solutions: *i)* make the self-generated sequences short, and *ii)* anneal the probability of using self-generated vs. ground-truth samples to 0, according to some schedule.

Despite its impressive empirical performance, Huszar et al (Huszár, 2015) show that SS is an inconsistent training strategy which pushes models towards memorising the distribution of symbols conditioned on their position in the sequence instead of on the prefix of preceding symbols. According to the authors, SS pays no attention to the content of the sequence prefix, and uses the hidden states to implement a simple counter which makes the model likely to recover from its own mistakes. Moreover, it is possible that the good performance of the model on image captioning datasets is either due to the algorithm not running until convergence, or to a lucky combination of factors including the model structure, early stopping,

random restarts, and the annealing schedule. The authors recommend adversarial training strategies as a much better choice for generative models.

Tang et al (Tang et al., 2016) study the the problem of NLG at particular contexts or situations. The authors focus on user review data due to its richness of context, sentiments and opinions expressed. They propose two approaches built on top of the encoder-decoder framework to generate user reviews as text sequences from user product contexts. In the first approach, *Contexts to Sequences*, the authors encode the product context information $\overrightarrow{C} = \{\overrightarrow{c}_i\}_{i=1,\ldots,K}$, where $\overrightarrow{c}_i$ denotes a type of context and K the number of context types, into a continuous semantic representation, which is fed into an LSTM decoder to generate text sequences. Despite promising results shown by the method, the authors consider that for long generated sequences the information from contexts is not propagated to distant words. In their second approach, *Gated Contexts to Sequences*, the authors add skip-connections to directly build the dependency between contexts h_C and each word when predicting the next word x_{t+1} in a sequence. When a new word in a sequence is generated, it does not only depend on the current hidden state h_t, but it also depends on the context representation h_C. Similar to the first model, the decoder is a vanilla recurrent neural network with LSTM unit.

Focusing on the same problem as Tang et al (Tang et al., 2016), Dong et al (Dong et al., 2017) propose *Attention Enhanced Attribute to Sequence Model*. The model learns to encode product attributes into vectors by means of an encoder network, and then generate reviews by conditioning on the encoded vectors inside a sequence decoder, and an attention mechanism (Bahdanau et al., 2014), (Xu et al., 2015) which learns soft alignments between the input attributes and the generated words. The product review generation problem is formally defined as follows. Given input attributes $a = (a_1, \ldots, a_{|a|})$, generate a product review $r = (y_1, \ldots, y_{|r|})$ which maximizes the conditional probability $p(r|a)$:

$$p(r|a) = \prod_{t=1}^{|r|} p(y_t|(y_1, \ldots, y_{t-1}), a) \qquad (6)$$

While the number of attributes $|a|$ is fixed for each product, the review text r is a sequence of variable length. In our experiments we use the two models proposed by Tang et al (Tang et al., 2016) and Dong et al (Dong et al., 2017) to generate use product reviews given the context information and the review text of each product in the Amazon dataset.

In addition to the already mentioned models, we also employ a pre-trained model released by Google, commonly referred to as Google LM (Jozefowicz et al., 2016). The model is an important contribution to the field of neural language modeling which emphasizes large scale recurrent neural network training. The model was trained on the One Billion Word Benchmark (Chelba et al., 2013), a publicly available dataset containing mainly news data and used as a reference standard for measuring the progress of statistical language modeling. The dataset includes 1 billion words in total with a vocabulary of 800,000 unique words. While for count based language models it is considered a medium-sized dataset, for neural network based language models the benchmark is regarded as a very large dataset. In terms of the model architecture, the GoogleLM model is a 2-layer LSTM neural network with 8,192 and respectively 1,024 hidden units in each layer, the largest Google was able to fit into GPU memory. The model uses Convolutional Neural Networks (CNNs) character embeddings as input, and makes predictions one character at a time, which presents the advantage that the model does not need to learn long-term dependencies in the data. We employ GoogleLM to generate sentences with a topic which identifies with the existing three categories (books, electronics and movies) present in the Amazon dataset we used.

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) represent a training methodology for generative models via an adversarial process, and are aimed at generating synthetic data which resembles the real data. The GAN framework works through the interplay between two feedforward neural network models, a generative model G and a discriminative model D, trained simultaneously by competing against each other. The generative model G aims to capture the data distribution and generate high quality synthetic data, while the discriminative model D estimates the probability a sample comes from the real training data and not from the synthetic data generated by G. Concretely, the generator G takes as input a vector of random numbers z, and transforms it into the form of the data we are interested

in imitating; the discriminator D takes as input either the real data x or generated data $G(z)$, and outputs probability $P(x)$ of the respective data being real. The GAN framework is equivalent to a minimax two-player game between the two models G and D:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{7}$$

Adversarial learning algorithms iteratively sample batches from the data and noise distributions, and use noisy gradient information to simultaneously ascend in the parameters θ_d of D, while descending in the parameters θ_g of G. The discriminator D is optimized to increase the likelihood of assigning a high probability to the real data x and a low probability to the fake generated data $G(z)$. The gradient for the discriminator can be expressed as follows:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(x^{(i)}) + \log(1 - D(G(z^{(i)}))) \right] \tag{8}$$

Alternatively, the generator G is optimized to increase the probability the generated data $G(z)$ is rated highly:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \left[\log(1 - D(G(z^{(i)}))) \right] \tag{9}$$

The goal of the generator G is to maximize the probability of discriminator D making a mistake by generating highly realistic data, while the discriminator D is learnt to distinguish whether a given data instance is real or not. The gradient of the training loss from the discriminator D is used as guidance for updating the parameters of the generator G. Gradient optimization is alternated between the two networks D and G as illustrated in Equations 8 and 9 on batches of real and generated data until GAN converges, at which point the data produced by GAN is the most realistic the network is capable of modeling.

However, GAN's applicability to discrete data is limited, despite the great success at generating realistic real valued synthetic samples in many computer vision tasks for eg., image generation (Brock et al., 2016), (Zhu et al., 2016), (Taigman et al., 2016), image style transfer (Luan et al., 2017), (Zhu et al., 2017) and semantic segmentation (Luc et al., 2016), (Souly et al., 2017). Training generative models of text using GANs is challenging due to the discrete nature of text data, which makes it difficult to backpropagate the gradient from the discriminator D to the generator G. GANs are designed for generating real-valued, continuous data, and the gradient of the loss from discriminator D w.r.t. the output of generator G is used to guide G to slightly change the generated value to make it more realistic (i.e. the gradient of the output of the discriminator network with respect to the synthetic data indicates how to slightly change the synthetic data to make it more plausible). Changes can be made to the synthetic data if it is based on real numbers, however for discrete tokens the slight change guidance is not a useful signal, as it is very likely that there is no corresponding token to the slight change given the limited vocabulary space[3]. In addition, a further reason why GANs cannot be applied to text data is because the discriminator D can only asses a complete sequence. When having to provide feedback for partially generated sequences, it is non-trivial to balance the current score of the partially generated sequence with the future score after the entire sequence has been generated (Yu et al., 2017). In the literature there are two approaches on how to deal with the problem of non-differentiable output and finding the optimal weights in a neural network: the REINFORCE algorithm, and Gumbel-Softmax reparameterization. We present each method below.

REINFORCE (Williams, 1992) algorithms, also known as *REward Increments, score-function estimators*, or *likelihood-ratio methods* adjust the weights of a neural network based on the log derivative trick in a direction that lies along the gradient of expected reinforcement without explicitly computing gradient estimates. It is a policy gradient method which uses the likelihood ratio trick $\left(\frac{\nabla_\theta p(X,\theta)}{P(X,\theta)} = \nabla_\theta \log p(X,\theta); \frac{\partial}{\partial_x} \log f(x) = \frac{f'(x)}{f(x)}\right)$ to update the parameters of an agent and increase the probability that the agent's policy will select a rewarding action given a state. Given the trajectory $\tau_t = (u_1, \ldots, u_{t-1}, x_0, \ldots, x_t)$ made up of a sequence of states x_k and control actions u_k, the goal of policy gradient is to find policy π_ϑ which takes

[3] https://www.reddit.com/r/MachineLearning/comments/40ldq6/generative_adversarial_networks_for_text/

as input trajectory τ_t and outputs a new control action that maximizes the total reward after L time steps. π_ϑ is a parametric randomized policy which assumes a probability distribution over actions:

$$p(\tau;\vartheta) = \prod_{t=0}^{L-1} p(x_{t+1}|x_t,u_t)\pi_v(u_t|\tau_t) \tag{10}$$

If we define the reward of a trajectory as:

$$R(\tau) = \sum_{t=0}^{N} R_t(x_t,u_t), \tag{11}$$

the reinforcement learning optimization problem becomes:

$$\max_\vartheta J(\vartheta) = \max_\vartheta \mathbb{E}_{p(\tau|\vartheta)}[R(\tau)] \tag{12}$$

Then policy gradient can be derived as follows:

$$\begin{aligned}\nabla_\vartheta J(\vartheta) &= \int R(\tau)\nabla_\vartheta p(\tau;\vartheta)d\tau \\ &= \int R(\tau)\frac{\nabla_\vartheta p(\tau;\vartheta)}{p(\tau;\vartheta)}p(\tau;\vartheta)d\tau \\ &= \int (R(\tau)\nabla_\vartheta \log p(\tau;\vartheta))p(\tau;\vartheta)d\tau \\ &= \mathbb{E}_{p(\tau;\vartheta)}[R(\tau)\nabla_\vartheta \log p(\tau;\vartheta)]\end{aligned} \tag{13}$$

From Equation 13 we have that the gradient of J w.r.t. ϑ is equal to the expected value of the function $G(\tau,\vartheta) = R(\tau)\nabla_\vartheta \log p(\tau;\vartheta)$. This function provides an unbiased estimate of the gradient of J and can be computed by running policy π_ϑ and sampling a trajectory τ without knowing the dynamics of the system since $p(x_{t+1}|x_t,u_t)$ does not depend on parameter ϑ. Following this direction is equivalent to running stochastic gradient descent on J.

$$\nabla_\vartheta \log p(\tau;\vartheta) = \sum_{t=0}^{L-1} \nabla_\vartheta \log \pi_\vartheta(u_t|\tau_t) \tag{14}$$

The policy gradient algorithm can be summarized:

1. Choose ϑ_0, stepsize sequence α_k, and set $k = 0$;
2. Run the simulator with policy π_{ϑ_k} and sample τ_k;
3. $\vartheta_{k+1} = \vartheta_k + \alpha_k R(\tau_k)\sum_{t=0}^{L-1}\nabla_\vartheta \log \pi_\vartheta(u_{tk}|\tau_t)$;
4. $k = k+1$, go to step 2.

The policy gradient algorithm can be run on any problem if sampling from π_ϑ can be done efficiently. Policy gradient is simple as it optimizes over a parametric family $p(u;\vartheta)$ instead of optimizing over the space of all probability distributions. However, there are constraints regarding the probability distribution, which should be easy to sample from, easy to search by gradient methods, and rich enough to approximate delta functions. In addition, the complexity of the method depends on the dimensionality of the search space and can be slow to converge. Finally, the policy gradient update is noisy, and its variance increases proportionally with the simulation length L.

The other solution to the problem of dealing with non-differentiable output is to use the the *Gumbel-Softmax* (Jang et al., 2016) approach, and replace the non-differentiable sample from the categorical distribution with a differentiable sample from a Gumbel-Softmax distribution. The Gumbel-Softmax distribution is a continuous distribution on the simplex that can approximate categorical samples. Parameter gradients can be easily computed by applying the reparameterization trick (Kingma and Welling, 2013), a popular technique used in variational inference and adversarial learning of generative models in which the expectation of a measurable function g of a random variable ϵ is calculated by integrating $g(\epsilon)$ with respect to the distribution of ϵ:

$$\mathbb{E}(g(\epsilon)) = \int g(\epsilon)dF_\epsilon \tag{15}$$

Therefore, in order to compute the expectation of $z = g(\epsilon)$ we do not need to know explicitly the distribution of z, but only know g and the distribution of ϵ. This can alternatively be expressed as:

$$\mathbb{E}_{\epsilon\sim p(\epsilon)}(g(\epsilon)) = \mathbb{E}_{z\sim p(z)}(z) \tag{16}$$

If the distribution of variable z depends on parameter ϕ, i.e. $z \sim p_\phi(z)$, and if we can assume $z = g(\epsilon,\phi)$ for a known function g of parameters ϕ and noise distribution $\epsilon \sim \mathcal{N}(0,1)$, then for

any measurable function f:

$$\begin{aligned}\mathbb{E}_{\epsilon\sim p(\epsilon)}(f(g(\epsilon,\phi))) &= \mathbb{E}_{z\sim p_\phi(z)}(f(z)) \\ \mathbb{E}_{\epsilon\sim p(\epsilon)}(\nabla f(g(\epsilon,\phi))) &= \nabla_\phi \mathbb{E}_{\epsilon\sim p(\epsilon)}(f(g(\epsilon,\phi))) \\ &= \nabla_\phi \mathbb{E}_{z\sim p_\phi(z)}(f(z))\end{aligned} \tag{17}$$

In equation 17, z has been conveniently expressed such that functions of z can be defined as integrals w.r.t. to a density that does not depend on the parameter ϕ. Constructing unbiased estimates of the gradient is done using Monte Carlo methods:

$$\nabla_\phi \mathbb{E}_{z\sim p_\phi(z)}(f(z)) \sim \frac{1}{M}\sum_{i=1}^{M} \nabla f(g(\epsilon^i,\phi)) \tag{18}$$

The reparameterization trick aims to make the randomness of a model an input to that model instead of letting it happen inside the model. Given this, the network model is deterministic and we can differentiate with respect to sampling from the model. An example of applying the reparameterization trick is to rewrite samples drawn from the normal distribution $z \sim \mathcal{N}(\mu,\sigma)$ as $z = \mu + \sigma\epsilon$, with $\epsilon \sim \mathcal{N}(0,1)$. In this way stochastic nodes are avoided during backpropagation. However, the reparameterization trick cannot be directly applied to discrete valued random variables, for eg. text data, as gradients cannot backpropagate through discrete nodes in the computational graph.

The Gumbel-Softmax trick attempts to overcome the inability to apply the reparameterization trick to discrete data. It parameterizes a discrete distribution in terms of a Gumbel distribution, i.e. even if the corresponding function is not continuous, it will be made continuous by applying a continuous approximation to it. A random variable G has a standard Gumbel distribution if $G = -\log(-\log(U)), U \sim \text{Unif}[0,1]$. Any discrete distribution can be parameterized in terms of Gumbel random variables as follows. If X is a discrete random variable with $P(X=k) \propto \alpha_k$ random variable and $\{G_k\}_{k\leq K}$ an i.i.d. sequence of standard Gumbel random variables, then:

$$X = \arg\max_k(\log\alpha_k + G_k) \tag{19}$$

Equation 19 illustrates sampling from a categorical distribution: draw Gumbel noise by transforming uniform samples, add it to $\log\alpha_k$, then take the value of k that yields the maximum. The $\arg\max$ operation that relates the Gumbel samples is not continuous, however discrete random variables can be expressed as one-hot vectors and take values in the probability simplex:

$$\Delta^{K-1} = \{x \in R_+^K, \sum_{k=1}^{K} x_k = 1\} \tag{20}$$

A one_hot vector corresponds to a discrete category, and since the $\arg\max$ function is not differentiable, a softmax function can be used instead as a continuous approximation of $\arg\max$:

$$f_\tau(x)_k = \frac{\exp(x_k/\tau)}{\sum_{k=1}^{K}\exp(x_k/\tau)} \tag{21}$$

Therefore, the sequence of simplex-valued random variables X^τ is:

$$\begin{aligned}X^\tau = (X_k^\tau)_k &= f_\tau(\log\alpha + G) \\ &= \frac{\exp((\log\alpha_k + G_k)/\tau)}{\sum_{i=1}^{K}\exp((\log\alpha_i + G_i)/\tau)}\end{aligned} \tag{22}$$

Equation 22 is known as the Gumbel-Softmax distribution and can be evaluated exactly for different values of x, α and τ, where τ is a temperature parameter that controls how closely the samples from the Gumbel-Softmax distribution approximate those from the categorical distribution. When $\tau \to 0$, the softmax function becomes an $\arg\max$ function and the Gumbel-Softmax distribution becomes the categorical distribution. At training time τ is a set to a value greater than 0 which allows gradients to backpropagate past the sample, and then is gradually annealed to a value close to 0. The Gumbel Softmax trick is important as it allows for the inference and generation of discrete objects. A direct application of this technique is generating text via GANs.

In summary, GANs have shown impressive performance at generating natural images nearly indistinguishable from real images, however applying GANs to text generation is a non-trivial task due to the special nature of the linguistic representation. According to Dai et al (Dai et al., 2017), the two main challenges to overcome when using GANs with textual input are:

i) first, text generation is a sequential non-differentiable sampling procedure which samples a discrete token at each time step (vs. image generation where the transformation from the input random vector to the produced output image

is a deterministic continuous mapping); the non-differentiability of text makes it difficult to apply back-propagation directly, and to this end, classical reinforcement learning methods such as Policy Gradient (Sutton et al., 2000) have been used. In policy gradient the production of each word is considered as an action for which the reward comes from the evaluator, and gradients can be back-propagated by approximating the stochastic policy with a parametric function.

ii) second, in the GAN setting the generator receives feedback from the evaluator when the entire sample is produced, however for sequence generation this causes difficulties during training, such as vanishing gradients and error propagation. To allow the generator to get early feedback when a text sequence is partly generated, Monte Carlo rollouts are used to calculate the approximated expected future reward. This has been found empirically to improve the efficiency and stability of the training process.

Unlike in conventional GAN settings that deal with image generation, the production of sentences is a discrete sampling process, which is also non-differentiable. A natural question that arises is how can the feedback be back-propagated from the discriminator to the generator under such a formulation. Policy gradient considers a sentence as a sequence of actions, where each word w_t is an action and the choices of such actions are governed by a policy π_θ. The generative procedure begins with an initial state $S_{1:0}$ which is the empty sentence, and at each time step t the policy π_θ takes as input the previously generated words $S_{1:t-1}$ up until time $t-1$, as well as the noise vector z, and yields a conditional distribution $\pi_\theta(w_t|z, S_{1:t-1})$ over the vocabulary words. The computation is done one step at a time moving along the LSTM network and sampling an action w_t from the conditional distribution up until w_t will be equal to the end of sentence indicator, in which case the sentence is terminated. The reward for the generated sequence of actions S will be a score r calculated by the discriminator. However, this score can be computed only after the sentence has been completely generated, and in practice this leads to difficulties such as vanishing gradients and very slow training convergence. Early feedback is used to evaluate the expected future reward when the sentence is partially generated, and the expectation can be approximated using Monte Carlo rollouts. The Monte Carlo rollout method is suitable to use when a part of the sentence $S_{1:t}$ has been already generated, and we continue to sample the remaining words of the sentence from the LSTM network until the end of sentence token is encountered. The conditional simulation is conducted n times, which results in n sentences. For each sentence we compute an evaluation score, and the rewards obtained by the simulated sentences are averaged to approximate the expected future reward of the current sentence. In this way updating the generator is possible with feedback coming from the discriminator. The utility of the policy gradient method is that by using the expected future reward the generator is provided with early feedback and becomes trainable with gradient descent.

Yu et al propose SeqGAN (Yu et al., 2017), a GAN-based sequence generation framework with policy gradient, which is the first work to employ GANs for generating sequences of discrete tokens to overcome the limitations of GANs on textual data. SeqGAN treats the sequence generation procedure as a sequential decision making process (Bachman and Precup, 2015). A discriminator is used to evaluate the generated sequence and provide feedback to the generative model to guide its learning. It is a well known problem of GANs that for text data (discrete ouputs) the gradient cannot be passed back from the discriminator to the generator. SeqGAN addresses this problem by treating the generator as a stochastic parameterized policy trained via policy gradient (Sutton et al., 2000) and optimized by directly performing gradient policy update, therefore avoiding the differentiation difficulty for discrete data. The reinforcement learning reward comes from the discriminator based on the likelihood that it would be fooled judged on a complete sequence of tokens, and is passed back to the intermediate state-action steps using Monte Carlo search (Browne et al., 2012).

The sequence generation problem is defined as follows. Given a dataset of human written sequences, train a generative model G_θ parameterized by θ to output sequence $Y_{1:T} = (y_1, \ldots, y_t, \ldots, y_T), y_t \in Y$, where Y is the word vocabulary. The current state is the sequence of tokens $(y_1, \ldots, y_{t-1})$ generated until timestep t, and the action a taken from this state is the selection of next token y_t. The policy model $G_\theta(y_t|Y_{1:t-1})$ is stochastic and will select an action according to the leant probability distribution of the input to-

kens. The state transition from the current state $s = Y_{1:t-1}$ to the next state $s^{'} = Y_{1:t}$ after choosing action $a = y$ is deterministic, i. e. $\delta^{a}_{s,s^{'}} = 1$ for next state $s^{'}$, and $\delta^{a}_{s,s^{''}} = 0$ for other next states $s^{''}$. The discriminative model $D_\phi(Y_{1:T})$ is used to guide the generator G_θ, and outputs a probability indicating how likely a sequence $Y_{1:T}$ produced by G_θ comes from real sequence data. D_ϕ is trained with both real and fake examples from the real sequence data and the synthetic data generated by G_θ. The objective of the generator model (policy) $G_\theta(y_y|Y_{1:t-1})$ is to maximize its expected end reward R_T which comes from the discriminator D_ϕ for a sequence which is generated starting from initial state s_0:

$$J(\theta) = \mathbb{E}[R_T|s_0, \theta] = \sum_{y_1 \in Y} G_\theta(y_1|s_0) Q^{G_\theta}_{D_\phi}(s_0, y_1) \tag{23}$$

The action-value function $Q^{G_\theta}_{D_\phi}(s, a)$ for a sequence represents the expected cumulative reward starting from state s, taking action a and then following policy G_θ. The action value function $Q^{G_\theta}_{D_\phi}(s, a)$ is calculated as the estimated probability (reward) the discriminator $D_\phi(Y^n_{1:T})$ assigns to the generated sample being real:

$$Q^{G_\theta}_{D_\phi}(a = y_T, s = Y_{1:T-1}) = D_\phi(Y^n_{1:T}) \tag{24}$$

In the GAN setup, the discriminator D_ϕ can only provide a reward at the end of a finished sequence. In order to evaluate the action-value function $Q^{G_\theta}_{D_\phi}(s, a)$ for an intermediate state s, Monte Carlo search with roll-out policy G_β (identical to the generator G_θ policy) is used to sample the unknown remaining $T - t$ tokens that result in a complete sentence. The roll-out policy G_β starts from the current state s and is run for N times to get an accurate assessment of the action-value function $Q^{G_\theta}_{D_\phi}(s, a)$ through a batch of N output samples, thus reducing the variance of the estimation:

$$\{Y^1_{1:T}, \dots, Y^N_{1:T}\} = MC^{G_\beta}(Y_{1:t}; N)$$

$$Q^{G_\theta}_{D_\phi}(a = y_t, s = Y_{1:t-1}) = \begin{cases} \frac{1}{N}\sum_{n=1}^{N} D_\phi(Y^n_{1:T}), \\ \text{if } Y^n_{1:T} \in MC^{G_\beta}(Y_{1:t;N}), t < T \\ D_\phi(Y_{1:t}), \text{if } t = T \end{cases} \tag{25}$$

The generator starts with random sampling at first, but once more realistic samples have been generated, the discriminator D_ϕ is updated (which will in turn improve the generator model iteratively):

$$\min_\phi -\mathbb{E}_{Y \sim p_{\text{data}}}[\log D_\phi(Y)] - \mathbb{E}_{Y \sim G_\theta}[\log(1 - D_\phi(Y))] \tag{26}$$

The generator G_θ is updated every time a new discriminator D_ϕ has been obtained. The gradient of the generator's objective function $J(\theta)$ w.r.t the generator's parameters θ is expressed as follows:

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{Y_{1:t-1} \sim G_\theta} \Bigg[\sum_{y_t \in Y} \nabla_\theta G_\theta(y_t|Y_{1:t-1}) \cdot \cdot Q^{G_\theta}_{D_\phi}(Y_{1:t-1}, y_t) \Bigg] \tag{27}$$

Expectation $\mathbb{E}$ can be approximated by sampling methods, and generator's parameters are updated:

$$\theta \leftarrow \theta + \alpha_h \nabla_\theta J(\theta), \text{where } \alpha_h - \text{learning rate} \tag{28}$$

In the initial stages of training, the generator G_θ is pre-trained via maximum likelihood estimation, and the discriminator D_ϕ is pre-trained via minimizing the cross-entropy between the ground truth label and the predicted probability; after the pre-training stage is over, the generator and the discriminator are trained alternatively. The SeqGAN authors chose an LSTM (Schmidhuber and Hochreiter, 1997) architecture for the generator in order to avoid the vanishing and the exploding gradient problem of back-propagation through time, and a CNN (LeCun et al., 1998), (Kim, 2014) architecture with highway networks (Srivastava et al., 2015) as discriminator. The evaluation metric is set to minimize the average negative log-likelihood between the generated data and an oracle considered as the human observer:

$$\text{NLL}_{\text{oracle}} = -\mathbb{E}_{Y_{1:T} \sim G_\theta} \Bigg[\sum_{t=1}^{T} \log G_{\text{oracle}}(y_t|Y_{1:t-1}) \Bigg] \tag{29}$$

Lin et al. (Lin et al., 2017) consider that GANs restrict the discriminator too much by forcing it to be a binary classifier. Because of this setup, the discriminator is limited in its learning capacity especially for tasks with a rich structure, such as when generating natural language expressions.

The authors propose a generative adversarial framework called RankGAN, which is able to capture the richness and diversity of language by learning a relative ranking model between the machine written and human written sentences in an adversarial framework. The adversarial network consists of two neural network models, a generator G_θ and a ranker R_ϕ, where θ and ϕ are parameters. The RankGAN discriminator R_ϕ, instead of performing a binary classification task as in conventional GANs, is trained to rank the machine-written sentences lower than human-written sentences w.r.t. a human-written reference set. Alternatively, the generator G_θ is trained to confuse the ranker R in such a way that machine written sentences are ranked higher than human written sentences with regard to the reference set. The authors consider that by viewing a set of samples collectively (instead of just one sample) and evaluating their quality through relative ranking, the discriminator can make better judgements regarding the quality of the samples, which helps in turn the generator better learn to generate realistic sequences. The problem can be expressed mathematically as G_θ and R_ϕ playing a minimax game with the objective function $\mathcal{L}$:

$$\min_\theta \max_\phi \mathcal{L}(G_\theta, R_\phi) = \mathbb{E}_{s\sim P_h}[\log R_\phi(s|U, C^-)] + \mathbb{E}_{s\sim G_\theta}[\log(1 - R_\phi(s|U, C^+))] \tag{30}$$

The ranker R_ϕ is optimized to increase the likelihood of assigning a high probability to the real sentence s and a low probability to the fake generated data G_θ. $s \sim P_h$ denotes that sentence s is sampled from human written sentences, while $s \sim G_\theta$ denotes that sentence s is sampled from machine written sentences. U is a reference set which is used for estimating relative ranks. C^+ and C^- are comparison sets with regards to input sentences. When the input sentence s is sampled from the real data, C^- is sampled from the generated data, and alternatively when the sentence s is sampled from the synthetic data generated by G_θ, C^+ is sampled from human written data.

Similar to SeqGAN, the authors use policy gradient to overcome the non-differentiability problem of text data. However, unlike SeqGAN, the regression based discriminator is replaced with a ranker and a new learning objective function. The generative model G_θ is an LSTM network, while the ranker R_ϕ is a CNN network. The rewards for training the model are encoded with relative ranking information. When a sequence is incomplete, an intermediate reward is computed using Monte Carlo rollout methods. The expected future reward V for partial sequences is defined as:

$$V_{\theta,\phi}(s_{1:t-1,U}) = \mathbb{E}_{s_r\sim G_\theta}[\log R_\phi(s_r|U, C^+, s_{1:t-1})] \tag{31}$$

In Equation 31 above, s_r denotes a complete sequence sampled by using rollout methods starting from sequence $s_{1:t-1}$. A total of n different paths are sampled, and their corresponding ranking scores are computed. The average ranking score is used to approximate the expected future reward for the current partially generated sequence $s_{1:t-1}$; the ranking score of an input sentence s given reference sentence u and comparison set C (where $C = C^+$ if sentence s is machine generated, $C = C^-$ otherwise) is computed using a softmax-like formula:

$$P(s|u, C) = \frac{\exp(\gamma\alpha(s|u))}{\sum_{s'\in C'}\exp(\gamma\alpha(s'|u))}, \text{where}$$
$$\alpha(s|u) = \cos(y_s, y_u) = \frac{y_s y_u}{||y_s||||y_u||} \tag{32}$$

In Equation 32, y_s is the embedded feature vector of the input sentence, and y_u is the embedded feature vector of the reference sentence. The gradient of the objective function for generator G_θ for start state s_0, vocabulary V, and generator policy π_θ is computed as:

$$\nabla_\theta \mathcal{L}_\theta(s_0) = \mathbb{E}_{s_{1:T}\sim G_\theta}\left[\sum_{t=1}^{T}\sum_{w_t\in V}\nabla_\theta\pi_\theta(w_t|s_{1:t-1})\cdot \cdot V_{\theta,\phi}(s_{1:t}, U)\right] \tag{33}$$

Therefore, RankGAN deals with the gradient vanishing problem of GAN by replacing the original binary classifier discriminator with a ranking model in a learning-to-rank framework. The ranking score is computed by taking a softmax over the expected cosine distances from the generated sequences to the real data.

Guo et al (Guo et al., 2018) find that a limitation of current GAN frameworks for text generation (Yu et al., 2017), (Lin et al., 2017), (Rajeswar et al., 2017), (Che et al., 2017), (Li et al., 2017),

(Zhang et al., 2017) is that they are only capable of generating short texts, within a limited length of around 20 tokens. Generating longer sequences is a less studied but more challenging research problem with a lot of useful applications, such as the auto-generation of news articles or product descriptions. Nevertheless, long text generation faces the issue that the binary guiding signal from generator D is sparse and non-informative; it does not provide useful information regarding the intermediate syntactic structure and semantics of the generated text so that the generator G could learn from that signal. Besides that, it is only available after the entire sequence has been generated, and the final reward value does not provide much guidance on how to alter the parameters of G at training time. Moreover, the approach of relying on binary feedback from the discriminator requires a very large number of real and generated samples to improve G. Aiming to make the guiding signal coming from the discriminator D more informative, the authors propose LeakGAN (Guo et al., 2018), a GAN approach for adversarial text generation in which the discriminative model D is allowed to leak its own high-level extracted features (in addition to providing the final reward value) to better guide the training of the generative model G. The authors pick a hierarchical generator for G, which is made up of two distinct modules: a *high-level manager* module, and a *low-level worker* module. The high level manager module (or mediator) receives the feature map representation of the discriminator D; this is not normally allowed in the conventional GAN setup as this feature map is internally maintained by the discriminator. The manager embeds this feature map representation coming from the discriminator and passes it over to the worker module. The worker first encodes the current generated sequence, and combines this resulting encoding with the embedding produced by the manager to decide what action to take at the current state. Therefore, LeakGAN "leaks" guiding signals from the discriminator D to the generator G more frequently and more informatively throughout the sequence generation process and not at the end only, helping G improve better and faster.

The discriminator D_ϕ is made up of a feature extractor $\mathcal{F}(.;\phi_f)$ and a final sigmoid classification layer. For input sequence s, D_ϕ is defined as:

$$D_\phi(s) = \text{sigmoid}(\phi_l^T \mathcal{F}(s;\phi_f)) = \text{sigmoid}(\phi_l^T f) \tag{34}$$

The feature vector in the last layer of D_ϕ is denoted as $f = \mathcal{F}(s;\phi_f)$, and it will be leaked to the generator G_θ. A natural implication of this approach is that the reward the generator G_θ receives for a partially generated sequence is directly related to the quality of the extracted features by the discriminator D_ϕ. Therefore, for the discriminator D_ϕ to yield a high reward, it is necessary to find a highly rewarding region in the extracted feature space. The authors consider that compared to a scalar signal, the feature vector f is more informative as it captures the position of the generated words in the extracted feature space. D_ϕ is implemented as a CNN network. The manager module $\mathcal{M}(f_t, h_{t-1}^M;\theta_m)$ of the hierarchical generator G_θ receives as input the extracted feature vector f_t, which it combines with its internal hidden state to produce the goal vector g_t:

$$\begin{aligned} g_t^{'} &= \mathcal{M}(f_t, h_{t-1}^M;\theta_m) \\ g_t &= \frac{g_t^{'}}{||g_t^{'}||} \end{aligned} \tag{35}$$

The goal vector embedding w_t of goal g_t is computed by applying a linear transformation ψ with weight matrix W_ψ to the sum of recent c goals:

$$w_t = \psi(\sum_{i=1}^{c} g_{t-i}) = W_\psi(\sum_{i=1}^{c} g_{t-i}) \tag{36}$$

w_t is fed to the worker module $\mathcal{W}(.;\theta_w)$, which is in charge with the generation of the next token. The worker module takes the current word x_t as input and outputs matrix O_t; this matrix is then combined through a softmax with the goal vector embedding w_t:

$$\begin{aligned} O_t, h_t^W &= \mathcal{W}(x_t, h_{t-1}^W;\theta_w) \\ G_\theta(.|s_t) &= \text{softmax}(O_t w_t/\alpha) \end{aligned} \tag{37}$$

At training time, the manager and the worker modules are trained separately – the manager is trained to predict which are the most rewarding positions in the discriminative feature space, while the worker

is rewarded to follow these directions. The gradient for the manager module is defined as:

$$\nabla^{\text{adv}}_{\theta_m} g_t = -Q_{\mathcal{F}}(s_t, g_t) \nabla_{\theta_m} d_{\cos}(f_{t+c} - f_t, g_t(\theta_m)) \tag{38}$$

$Q_{\mathcal{F}}(s_t, g_t)$ defines the expected reward under the current policy and can be approximated using Monte Carlo search. $d_{\cos}$ computes cosine similarity between the goal vector $g_t(\theta_m)$ produced by the manager, and the change in feature representation $f_{t+c} - f_t$ after c transitions. In order to achieve a high reward, the loss function is trying to force the goal vector to match the transition in feature space. Before the adversarial training takes place, the manager undergoes a pre-training stage with a separate training scheme which mimics the transition of real text samples in the feature space:

$$\nabla^{\text{pre}}_{\theta_m} = -\nabla_{\theta_m} d_{\cos}(f^{'}_{t+c} - f^{'}_t, g_t(\theta_m)) \tag{39}$$

The worker uses the REINFORCE algorithm during training to maximize the reward when taking action x_t given the previous state is s_{t-1}:

$$\nabla_{\theta_w} \mathbb{E}_{s_{t-1} \sim G} \left[\sum_{x_t} r^I_t \mathcal{W}(x_t | s_{t-1}; \theta_w) \right] =$$
$$\mathbb{E}_{s_{t-1} \sim G, x_t \sim \mathcal{W}(x_t | s_{t-1})} \left[r^I_t \nabla_{\theta_w} \log \mathcal{W}(x_t | s_{t-1}; \theta_w) \right]$$
$$r^I_t = \frac{1}{c} \sum_{i=1}^{c} d_{\cos}(f_t - f_{t-i}, g_{t-i}) \tag{40}$$

During the adversarial training process, the generator G_θ and the discriminator D_ϕ are trained in alternative stages. When the generator G_θ is trained, the worker $\mathcal{W}(.; \theta_w)$ and the manager $\mathcal{M}(.; \theta_m)$ modules are trained alternatively fixing each other.

Mode collapse (Goodfellow, 2016) is a common problem when training GAN models, when the generator learns to produce samples with extremely low variety, limiting the usefulness of the leant GAN model. In mode collapse the generator network learns to output samples from a few modes of the data distribution only, missing out on many other modes even though samples from these missing modes can be found throughout the training data. Mode collapse can range from complete collapse, when the generated samples are entirely identical, to partial collapse when the generated samples present some common properties (Srivastava et al., 2017), (Salimans et al., 2016). Several attempts have been made to address the problem, which include: *i)* directly encouraging the generator cost function to account for the diversity of the generated batches by comparing these samples across a batch in order to determine whether the entire batch is real or fake, *ii)* anticipate counterplay, in which the generator learns to fool the discriminator before the discriminator has a chance to respond (and therefore taking counterplay into account), *iii)* experience replay, which minimizes the switching between modes by showing old fake generated samples to the discriminator every now and then, and *iv)* using multiple GANs, in which a GAN is trained for each different mode so that when combined, the GANs altogether cover all modes.

In LeakGAN, in order to address mode collapse, the authors propose an interleaved training scheme, which combines supervised training using maximum likelihood estimation with GAN adversarial training (instead of carrying only GAN adversarial training after the pretraining stage). Blending two training schemes is considered useful by the authors as it helps LeakGAN overcome local minimums, alleviates mode collapse and acts as an implicit regularizer on the generative model.

A.2 Samples produced by the review generators

Figure 8 shows the instructions given to the AMT workers who participated in this study. In Figure 9 we include a screen-shot of the user interface when annotating reviews.

In what follows we present samples generated by the review generators on which human annotators disagree most on whether these are human-written or machine-generated.

- Word LSTM temp 1.0
  a) i so enjoyed this book . i felt __ though . i especially like loving horses in the __ . and the story is well written .
  b) one of a different type on locked paranormal / vacation book . i enjoyed the characters and the plot . great mixture of historical fiction .
  c) this first edition of the complete series 8 years over six episodes just makes you laugh . the original tv is by far my cup of tea !

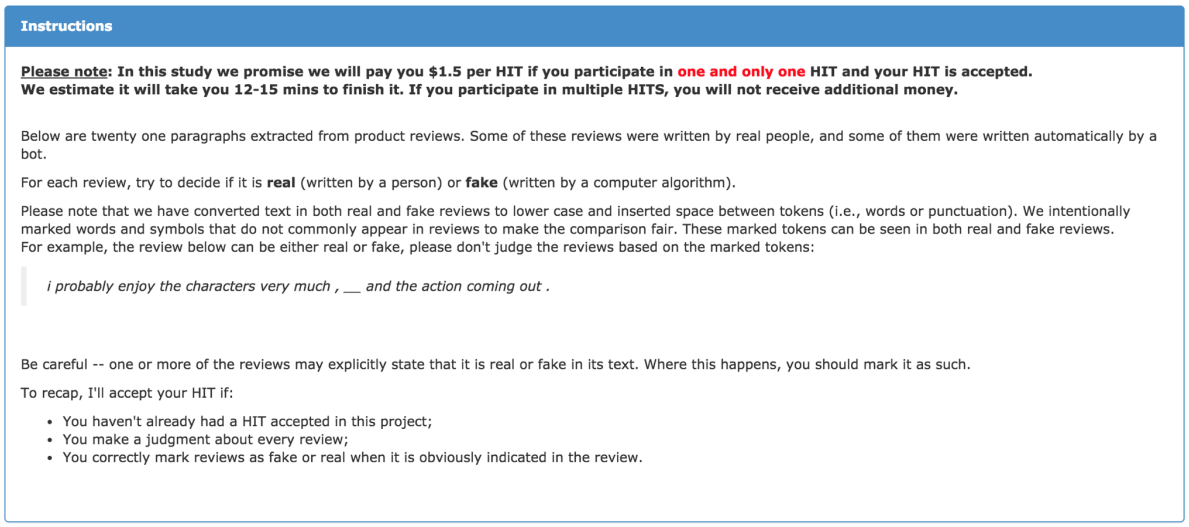

Figure 8: Screenshot of the instructions presented to Amazon Mechanical Turk workers.

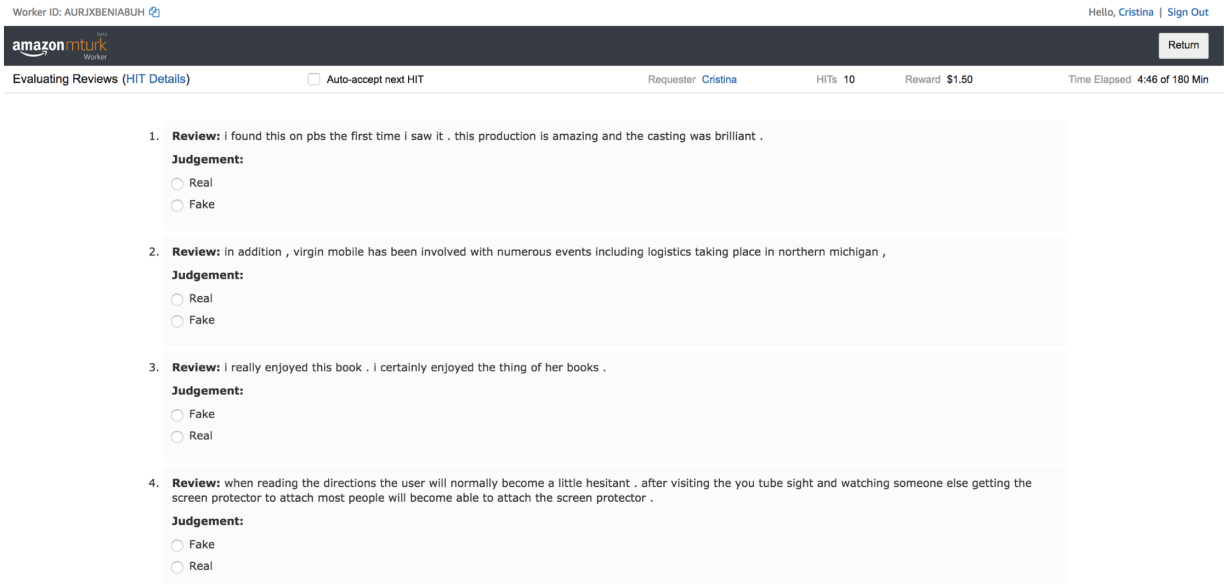

Figure 9: Screenshot of the Amazon Mechanical Turk user study interface.

d) works out of the box ! wouldn ' t spend the money for a better keyboard . use this with the matching kindle screen as well .

- Word LSTM temp 0.7

a) i am looking forward to the next book . i am a __ __ and i enjoyed the story . i like books where the characters are real .

b) this is an exciting book i could n ' t put down . i will probably read more books by this author . this is a must read .

c) okay , that ' s how i expected this movie . it was okay but it was so boring . i was bored and was disappointed .

d) this cable is not bad . it is so cheap and it works great . i ' ve used this for a couple of months now and __ on the ipad

- Word LSTM temp 0.5

a) this book was a great read ! the story was exciting and a bit __ . i really enjoyed the characters and the story line .

b) this is a great cable for the price . i would recommend this product to anyone needing a cable for a great price .

c) this is a great series . it is a must see for anyone who loves period dramas . i love the __ .

d) these batteries seem to be working as expected . i have had no problems with this product . i would recommend this to anyone .

- Scheduled Sampling

a) like most of the ones i have ! the tablet that came starts working properly .

b) i have had any almost using keyboards with an iphone case and kept it nicely and time . and it works well .

c) have got to watch it many times again and the seasons of __ each episode we can all watch it .

d) very interesting characters and likable characters that grow when you gave me __ of the __ because of the dog . what can i say is i absolutely loved it .

- Google LM

a) __ systems generally require less bandwidth and __ with operating systems , __ users to write and edit data nearly anywhere .

b) seems all but impossible to access . __ is all a __ and gets a bad __ on every __ .

c) __ is based in __ __ , __ , with a commercial office in __

d) oved this clip and the __ and __ apps were about so much fun that __ paid a big price . __ 2 and 3 like crazy .

- Attention Attribute to Sequence

a) i am always waiting for the next book to come out . i am a big fan of sean black and will .

b) purchased this to use with my macbook pro . it worked out perfectly , as described . no complaints .

c) great book all of the great mystery books . i enjoyed all of them and was sad when the book ended .

d) this is a great product . i ' ve had it for over a year now and it ' s still going strong . i ' m very happy with this purchase .

- Contexts to Sequences

a) i love this series . i love the characters and the story . i love the characters and the story line .

b) a great book and a great read . i love the characters and the story . i would recommend this book to anyone .

c) i enjoyed the story . it was a good read . i would recommend it to anyone who likes a good read .

d) i love this book and i love the characters . i love this book and i was not disappointed .

- Gated Contexts to Sequences

a) this is the first book i have read by this author . would recommend to anyone who likes a good romance book .

b) one of the best books i have ever read . the chemistry between the two main characters was a good read .

c) this book is awesome . lots of action and intrigue . i ' m glad i bought this book . thank you for sharing

d) great story and plot . sometimes a little slow at times but overall a good read .

- MLE SeqGAN

a) you will like this movie - get this set ... better than expected award for the characters . bad ending .

b) this switch converter works fine with all games and works perfect , sturdy program to zero manual products . nice feel .

c) i could not put it down . it was an interesting clean book , but i was expecting many more individuals in this story so i read in a long time .

d) great story . in college kids has been lost the __ mysteries , chris __ son is not better .

- SeqGAN

a) it was slow he kept me interested , and i think i thoroughly enjoyed the story .

b) i enjoyed this book and look forward to getting to __ larson .

c) received in excellent condition . i thought it was great but didn ' t know that movies were more than high ratings which i am my cup of tea .

d) awesome cute story . kudos to mr much __ of the sookie ' s story .

- RankGAN

a) robin williams is ok . just a great movie with __ now . __ is a great film with three stars ! wonderful video for a very good movie .

b) i have loved this movie so i could like the dvd sort of info . hot slow . love the old ford shows to though . __ a great actor .

c) this was a very amazing . __ laws and oh fact she became __ and __ is very unlikely together on the case .

d) i say so i would that originally arrived so i love the circular inch screen . i am sad how it works .

- LeakGAN

a) i really enjoyed reading this book . the author did an excellent job in delivering for all his writing books into us as business . a great summer read .

b) just loved it , so much could read more of this series , i like it but it was not written in a book that is well written , but very interesting .

c) i love hockey - baseball movie coming meets hockey ' s et addicted fear the birds feature so popular films have developed far worse reviews .

d) a very good book with a lot of twists in this book . i will be checking out more of this author next book .

A.3 Results

A.3.1 Human Evaluators

We chose the task of distinguishing machine-generated from real reviews because it is a straightforward surrogate of a Turing test. Moreover, how much their generated content can fool humans has been a key claim of many artificial intelligence models recently. The low inter-rater agreement suggests that this is a difficult task even for humans, which we hope would trigger the community to rethink about these claims. There are indeed finer-grained, perhaps more agreeable aspects of text quality (including semantic coherence, syntactic correctness, fluency, adequacy, diversity and readability). We decided not to include them in this experiment for two reasons: 1) as the first study, we are not sure which aspects human raters would consider when they judge for the realism of a review; 2) we wanted to keep the experiment design simple, and many of these aspects are harder to define. In the post-experiment survey, the raters commented on the reasons why they considered reviews as fake.

The low inter-rater agreement (0.27) reflects the difficulty/ subjectivity of the task: identifying individual reviews as human-written or machine-generated. Low human agreement is commonly reported in subjective evaluation tasks. Since our goal is to evaluate the **evaluators** instead of the competing algorithms, it is important to use a task neither too easy or too hard, so that there are distinguishable differences among the performances of competitors (including humans). When using the majority vote of human judgements, the accuracy of humans improved to a reasonable 72.63 %.

A.3.2 Discriminative Evaluators

In Table 3 and Table 4 we present comprehensive results for the meta-adversarial evaluators.

A.3.3 Text-Overlap Evaluators

In Figure 10 we present detailed results for all word overlap evaluators we used in this study.

A.3.4 Comparing Evaluators

In Table 5 we present correlation results between the evaluators included in this work.

A.3.5 Diversity Analysis

In Table 6 we present results for the Self-BLEU metric, while in Table 7 we present the correlation of Self-BLEU with the other evaluators. In addition, in Table 8 we present correlation results for BLEU G-Train and the rest of the evaluators.

B Discussion

B.1 User Study

A more detailed list of major clusters of reasons is as follows:

1. Grammar/ typo/ mis-spelling: the language does not flow well.
2. Too general/ too generic/ vagueness: generated reviews are vague, in lack of details.
3. Word choice (wording): in lack of slang, use the wrong words.
4. Flow (not fluent)/ structured/ logical: the sentences level language errors.
5. Contradictory arguments: some arguments support opposite opinions.
6. Emotion: lack of emotion, personality in the comments.
7. Repeated text: using words/ phrases repetitively.
8. Overly same as human: too advertisement, too formal, too likely to be real.

Table 3: Accuracy of deep (LSTM) and shallow (SVM) meta-adversarial evaluators. **The lower the better.** Meta-adversarial evaluators do better than humans on individual reviews, with less bias between the two classes. GAN-based generators are considered to be the best by meta-adversarial evaluators.

Generators	LSTM	SVM
Word LSTM temp 1.0	48.29 %	**50.31** %
Word LSTM temp 0.7	92.58 %	78.69 %
Word LSTM temp 0.5	99.31 %	94.74 %
Scheduled Sampling	50.09 %	51.31 %
Google LM	84.58 %	78.59 %
Attention Attribute to Sequence	90.08 %	74.37 %
Contexts to Sequences	100.00 %	100.00 %
Gated Contexts to Sequences	98.37 %	96.26 %
MLE SeqGAN	**41.45** %	52.35 %
SeqGAN	50.05 %	56.20 %
RankGAN	66.28 %	70.17 %
LeakGAN	87.03 %	77.55 %
D-test (all)	77.58 %	74.50 %
D-test (human-written)	80.12 %	75.98 %
D-test (machine-generated)	75.04 %	73.01 %

B.2 Granularity of Judgements

We charged the Turkers to label individual reviews as either fake or real. Each human judge only annotates 20 reviews, and they do not know which reviews are generated by the same generator. Comparing to an adversarial discriminator, a human judge has not seen many "training" examples of fake reviews or generators. That explains why the meta-adversarial evaluators are better at identifying fake reviews. In this context, humans are likely to judge whether a review is real based on how "similar" it appears to the true reviews they are used to see online. That is probably why their decisions are better correlated to text-overlap metrics that measures the similarity between a review and a set of references. This hypothesis is supported by a post-experiment survey of the human judges. Please see Appendix A.2 for user study samples.

This finding provides interesting implications to the selection of evaluation methods for different tasks. In tasks that are set up to judge individual pieces of generated text (e.g., reviews, translations, summaries, captions, fake news) where there exists human-written ground-truth, it is better to use word-overlap metrics instead of adversarial evaluators. Indeed, when the audience are not trained by reading lots of bot-generated texts, it is more reasonable to use an evaluator that mimics their decision-making process.

In some scenarios, the task is to make judgments in the context of a longer conversation or a set of documents (e.g., conversation agents, dialogue systems, social bots). The difference is that human subjects are exposed to machine-generated text, so that they may be better trained to distinguish fake from real. Moreover, when judgments are made on the agent/ system level (e.g., whether a Twitter account is a bot), signals like how similar the agent

Table 4: Accuracy of deep (LSTM, CNN, CNN & LSTM) and shallow (SVM, RF, NB, XGBoost) meta-adversarial evaluators. **The lower the better.** Meta-adversarial evaluators do better than humans on individual reviews, with less bias between the two classes. GAN-based generators are considered best by meta-adversarial evaluators.

Generators	LSTM	CNN	CNN & LSTM	SVM	RF	NB	XGBoost
Word LSTM temp 1.0	48.29 %	55.22 %	45.68 %	**50.31** %	53.63 %	32.77 %	48.97 %
Word LSTM temp 0.7	92.58 %	93.14 %	91.02 %	78.69 %	81.05 %	79.92 %	80.49 %
Word LSTM temp 0.5	99.31 %	99.35 %	99.08 %	94.74 %	94.29 %	96.86 %	94.71 %
Scheduled Sampling	50.09 %	48.77 %	43.37 %	51.31 %	52.88 %	**20.97** %	44.12 %
Google LM	84.58 %	74.03 %	74.85 %	78.59 %	82.71 %	48.28 %	82.41 %
Attention Attribute to Sequence	90.08 %	91.78 %	89.94 %	74.37 %	77.29 %	80.02 %	71.68 %
Contexts to Sequences	100.00 %	100.00 %	99.97 %	100.00 %	99.98 %	100.00 %	99.98 %
Gated Contexts to Sequences	98.37 %	99.06 %	98.38 %	96.26 %	95.35 %	98.63 %	93.62 %
MLE SeqGAN	**41.45** %	**47.54** %	**41.91** %	52.35 %	**51.14** %	21.83 %	**43.71** %
SeqGAN	50.05 %	52.91 %	47.35 %	56.20 %	54.91 %	25.60 %	48.11 %
RankGAN	66.28 %	67.23 %	59.37 %	70.17 %	61.94 %	35.98 %	61.23 %
LeakGAN	87.03 %	80.28 %	79.57 %	77.55 %	67.74 %	46.80 %	63.80 %
D-test (all)	77.58 %	74.72 %	75.18 %	74.50 %	70.31 %	70.74 %	73.79 %
D-test (human-written)	80.12 %	73.54 %	77.99 %	75.98 %	68.59 %	83.53 %	79.10 %
D-test (machine-generated)	75.04 %	75.90 %	72.38 %	73.01 %	72.04 %	57.95 %	68.48 %

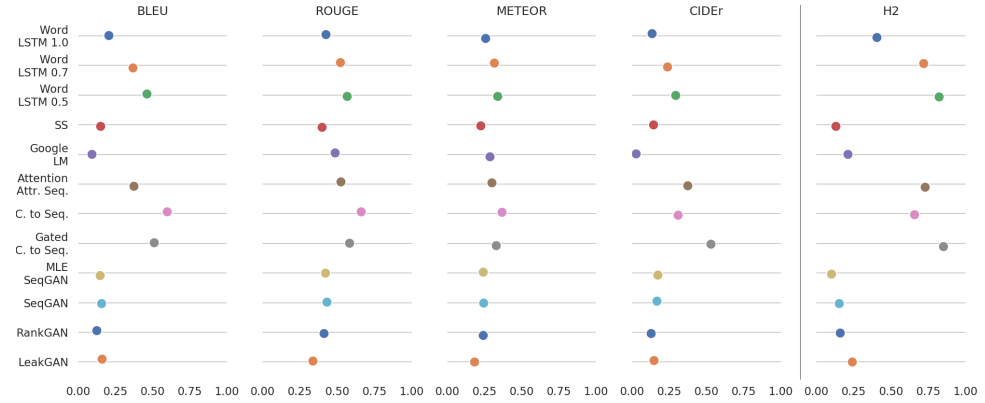

Figure 10: Text-Overlap Evaluators (BLEU, ROUGE, METEOR and CIDEr) scores for individual generators. **The higher the better.** The rankings are overall similar, as GAN-based generators are ranked low.

outputs are or how much the agent memorizes the training examples may become more useful than word usage, and a discriminative evaluator may be more effective than text-overlap metrics.

Our experiment also provide implications to improving NLG models, which implies that adversarial accuracy might not be the optimal objective for NLG if the goal is to generate documents that humans consider as real. Indeed, a fake review that fools humans does not necessarily need to fool a machine that has seen everything.

In contrast, GAN based models may perform better when judged as a whole system instead of individual items, or in a conversational context. When the human judges have seen enough examples from the same generator, the next example had better be somewhat different.

B.3 Imperfect Ground-truth

One important thing to note is that all discriminative evaluators are trained using natural labels (i.e., treating all examples from the Amazon review dataset as positive and examples generated by the candidate models as negative) instead of human-annotated labels. It is possible that if they were trained with human labels, the discriminative evaluators would have been more consistent to the human evaluators. Indeed, some reviews posted on Amazon may have been generated by bots, and if that is the case, treating them as human-written examples may bias the discriminators.

One way to verify this is to consider an alternative "ground-truth". We apply the already trained meta-discriminators to the human-annotated subset (3,600 reviews) instead of the full *D-test* set, and we use the majority vote of human judges (whether a review is fake or real) to surrogate the "ground-truth" labels (whether a review is generated or sampled from Amazon).

Surprisingly, when the meta-adversarial evaluators are tested using human majority-votes as ground-truth, both the accuracy numbers and the

Evaluation Method	Kendall tau-b (H1)	Spearman (H1)	Pearson (H1)	Kendall tau-b (H2)	Spearman (H2)	Pearson (H2)
SVM Individual-discriminators	-0.4545*	-0.6294*	-0.6716*	-0.5455*	-0.6783*	-0.6823*
LSTM meta-discriminator	-0.5455*	-0.7552*	-0.7699*	-0.6364*	-0.8042*	-0.7829*
CNN meta-discriminator	-0.6363*	-0.8112*	-0.8616*	-0.7273*	-0.8741*	-0.8766*
CNN & LSTM meta-discriminator	-0.6060*	-0.7902*	-0.8392*	-0.6970*	-0.8462*	-0.8507*
SVM meta-discriminator	-0.4545*	-0.6573*	-0.7207*	-0.5455*	-0.6993*	-0.7405
RF meta-discriminator	-0.5455*	-0.7273*	-0.7994*	-0.6364*	-0.7832*	-0.8075*
NB meta-discriminator	-0.6364*	-0.8112*	-0.9290*	-0.7273*	-0.8741*	-0.9388*
XGBoost meta-discriminator	-0.5455*	-0.7413*	-0.7764*	-0.6364*	-0.8042*	-0.7878*
BLEU evaluator	0.7576*	0.8601*	0.8974*	0.6666*	0.8182*	0.9060*
ROUGE evaluator	0.6060*	0.7692*	0.8054*	0.5758*	0.7483*	0.8073*
METEOR evaluator	0.5758*	0.7762*	0.8225*	0.5455*	0.7622*	0.8231*
CIDEr evaluator	0.5455*	0.7413*	0.8117*	0.4545*	0.6643*	0.8203*

Table 5: Kendall tau-b, Spearman and Pearson correlation coefficients between human evaluators $H1$, $H2$, and discriminative evaluators and word-overlap evaluators (* denotes statistical significant result with $p \leq 0.05$).

Generative Text Model	Self-BLEU	Lexical diversity
Word LSTM temp 1.0	0.1886	0.6467
Word LSTM temp 0.7	0.4804	0.2932
Word LSTM temp 0.5	0.6960	0.1347
Scheduled Sampling	0.1233	0.7652
Google LM	0.1706	**0.7745**
Attention Attribute to Sequence	0.5021	0.2939
Contexts to Sequences	0.8950	0.0032
Gated Contexts to Sequences	0.7330	0.1129
MLE SeqGAN	0.1206	0.7622
SeqGAN	0.1370	0.7330
RankGAN	**0.1195**	0.7519
LeakGAN	0.1775	0.7541

Table 6: Self-BLEU diversity scores per generator (the lower the more diverse), and lexical diversity scores (the higher the more diverse). There is high correlation between the two metrics with respect to the rankings of the generative text models.

Self-BLEU	Kendall tau-b	Spearman	Pearson
H1 evaluator	-0.8788*	-0.9301*	-0.8920*
H2 evaluator	-0.7879*	-0.8881*	-0.9001*
LSTM meta-discriminator	0.6667*	0.8252*	0.7953*
CNN meta-discriminator	0.7576*	0.8811*	0.8740*
CNN & LSTM meta-discriminator	0.7273*	0.8601*	0.8622*
SVM meta-discriminator	0.5758*	0.7413*	0.8518*
RF meta-discriminator	0.6667*	0.8112*	0.8944*
NB meta-discriminator	0.7576*	0.8811*	0.9569*
XGBoost meta-discriminator	0.6667*	0.8252*	0.8693*
BLEU evaluator	-0.8788	-0.9301*	-0.9880*
ROUGE evaluator	-0.7273*	-0.8392*	-0.9299*
METEOR evaluator	-0.6967*	-0.8462*	-0.8955*
CIDEr evaluator	-0.5455*	-0.7413*	-0.7987*

Table 7: Kendall tau-b, Spearman and Pearson correlation coefficients between Self-BLEU diversity rankings and the three evaluation methods - human evaluators $H1$, $H2$, discriminative evaluators and word-overlap based evaluators (* denotes statistical significant result with $p \leq 0.05$). Meta-discriminators have been trained on D-train, D-valid sets and tested on the **annotated D-test set with ground-truth test labels**.

BLEU G-train	Kendall tau-b	Spearman	Pearson
H1 evaluator	0.7176*	0.8511*	0.9111*
H2 evaluator	0.6260*	0.8091*	0.9209*
LSTM meta-discriminator	-0.5649*	-0.7461*	-0.7091*
CNN meta-discriminator	-0.6565	-0.7951*	-0.8213*
CNN & LSTM meta-discriminator	-0.6260*	-0.7811*	-0.7951*
SVM meta-discriminator	-0.4428*	-0.6130*	-0.7442*
RF meta-discriminator	-0.5038*	-0.6340*	-0.7864*
NB meta-discriminator	-0.6260*	-0.7601*	-0.9164*
XGBoost meta-discriminator	-0.5649*	-0.6550*	-0.7586*
BLEU evaluator	0.9619*	0.9912*	0.9936*
ROUGE evaluator	0.5954*	0.7496*	0.8717*
METEOR evaluator	0.6260*	0.7636*	0.8477*
CIDEr evaluator	0.6565*	0.8371*	0.8318*

Table 8: Kendall tau-b, Spearman and Pearson correlation coefficients between BLEU G-train rankings and the three evaluation methods - human evaluators $H1$, $H2$, discriminative evaluators and word-overlap based evaluators (* denotes statistical significant result with $p \leq 0.05$). Meta-discriminators have been trained on D-train, D-valid sets and tested on the **annotated D-test set with ground-truth test labels**.

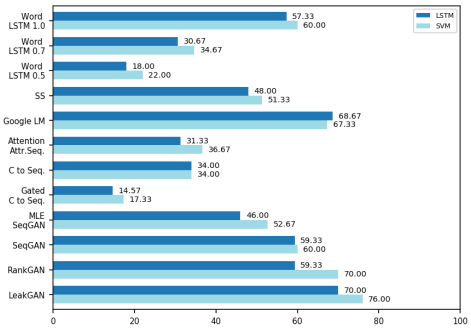

Figure 11: Accuracy of deep (LSTM) and shallow (SVM) meta-discriminators when tested on the **annotated subset of *D-test***, with *majority votes* as ground-truth. The lower the better.

rankings of the generators are significantly different from Table 3 and Table 4 (which used natural labels as ground-truth). We note that the scores

and rankings are more inline with the human evaluators. To confirm the intuition, we calculate the correlations between the meta-discriminators and the human evaluators using the annotated subset only. Replacing the natural ground-truth with human annotated labels, the meta-discriminators become positively correlated with human evaluators (Figure 6), although BLEU still appears to be the best evaluator.

These results indicate that when the "ground-truth" used by an automated Turing test is questionable, the decisions of the evaluators may be biased. Discriminative evaluators suffer the most from the bias, as they were directly trained using the imperfect ground-truth. Text-overlap evaluators are more robust, as they only take the most relevant parts of the test set as references (more likely to be high quality).

Our results also suggest that when adversarial training is used, the selection of training examples must be done with caution. If the "ground-truth" is hijacked by low quality or "fake" examples, models trained by GAN may be significantly biased. This finding is related to the recent literature of the robustness and security of machine learning models.

B.4 Role of Diversity

We also assess the role diversity plays in the rankings of the generators. To this end, we measure lexical diversity (Bache et al., 2013) of the samples produced by each generator as the ratio of unique tokens to the total number of tokens. We compute in turn lexical diversity for unigrams, bigrams and trigrams, and observe that the generators that produce the least diverse samples are easily distinguished by the meta-discriminators, while they confuse human evaluators the most. Alternatively, samples produced by the most diverse generators are hardest to distinguish by the meta-discriminators, while human evaluators present higher accuracy at classifying them. As reported in (Kannan and Vinyals, 2017), the lack of lexical richness can be a weakness of the generators, making them easily detected by a machine learning classifier. Meanwhile, a discriminator's preference for rarer language does not necessarily mean it is favouring higher quality reviews.

In addition to lexical diversity, Self-BLEU (Zhu et al., 2018) is an interesting measurement of the diversity of a set of text (average BLEU score of each document using the same collection as reference, therefore the lower the more diverse). In Figure 7 we present Self-BLEU scores for each generator, applied to their generated text in *D-test fake*. We also compute the correlation coefficients between the rankings of generators by Self-BLEU and the rankings by the evaluators (please see Figure 12). Results obtained indicate that Self-BLEU presents negative correlation with human evaluators and word-overlap evaluators, and positive correlation with discriminative evaluators. This result confirms the findings in literature (Kannan and Vinyals, 2017) that discriminators in adversarial evaluation are capturing known limitations of the generative models such as lack of diversity.

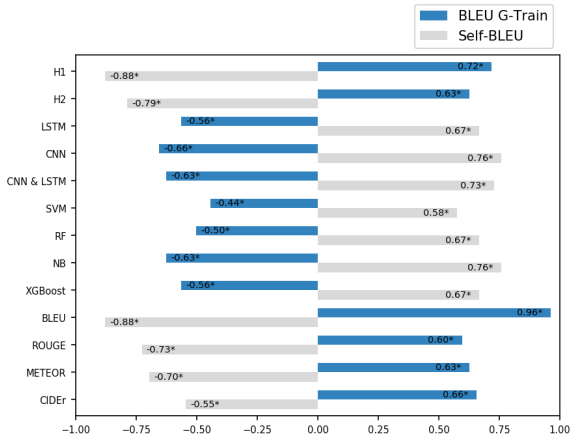

Figure 12: Kendall τ-b correlation coefficients between BLEU G-train and Self-BLEU rankings, and the three evaluation methods - human evaluators $H1$, $H2$, discriminative evaluators and word-overlap based evaluators (* denotes $p \leq 0.05$). Meta-discriminators have been trained on D-train, D-valid sets and tested on the **annotated D-test set with ground-truth test labels**.

Following this insight, an important question to answer is to what extent the generators are simply memorizing the training set *G-train*. To this end, we assess the degree of n-gram overlap between the generated reviews and the training reviews using the BLEU evaluator. In Table 9 we present the average BLEU scores of generated reviews using their nearest neighbors in *G-train* as references. We observe that generally the generators do not memorize the training set, and GAN models generate reviews that have fewer overlap with *G-train*. In Figure 12 we include the correlation between the divergence from training and the ratings by evaluators in the study. BLEU w.r.t. *G-train* presents highly positive correlation with BLEU w.r.t. *D-test real*, and it is also positively correlated with the

Generative Text Model	BLEU G-Train
Word LSTM temp 1.0	0.2701
Word LSTM temp 0.7	0.4998
Word LSTM temp 0.5	0.6294
Scheduled Sampling	0.1707
Google LM	0.0475
Attention Attribute to Sequence	0.5122
Contexts to Sequences	0.7542
Gated Contexts to Sequences	0.6240
MLE SeqGAN	0.1707
SeqGAN	0.1751
RankGAN	0.1525
LeakGAN	0.1871

Table 9: BLEU results when evaluating the generated reviews using G-train as the reference corpus (a lower score indicates less n-grams in common between the training set G-train and the generated text). GAN models present low similarity with the training set.

human evaluators $H1$ and $H2$.

The effects of diversity is perhaps not hard to explain. At the particular task of distinguishing fake reviews from real, all decisions are made on individual reviews. And because a human judge was not exposed to many fake reviews generated by the same generator, whether or not a fake review is sufficiently different from the other generated reviews is not a major factor for their decision. Instead, the major factor is whether the generated review looks similar to the reviews they have seen in reality. Instead, a discriminative evaluator makes the decision after seeing many positive and negative examples, and a fake review that can fool an adversarial classifier has to be sufficiently different from all other fake reviews it has encountered (therefore diversity of a generator is a major indicator of its ability to pass an adversarial judge).