Generating Yelp Reviews with GANs
Abhi Kulgod, Veeral Patel, Jayen Ram

**Motivation**
- Millions of users turn to Yelp as de facto site for choosing restaurants.
- Because of this popularity, there is a monetary incentive for both good and bad actors to detect or generate bad reviews.
- Good actor: Yelp filters out 25 percent of all submitted reviews it thinks are fake, biased, or unhelpful rants and raves.
- This preserves integrity of their site.
- Bad actor: New restaurant may generate fake reviews to show a false sense of quality and obtain more customers.
- Our generative adversarial network will tackle both of these problems. We will focus on the task of generating fake Yelp reviews since it’s a much complex problem (especially for adversarial training).

**Problem Setup**
- Datasets of real and suspected fake Yelp reviews are used for adversarial training.
- Limit generated review to 30 words and use massive dataset to help generate.
- For this problem, we want to generate reviews that won’t sound like fake, generic bots; using adversarial training will allow us to optimize for sentence generation as opposed to prediction using normal RNNs.

**Overview**
- Model the Generator as an RL agent where:
  - States and actions are represented by generated sequences and tokens.
  - Reward is modeled by the Discriminator.
- We first learn word embeddings using Word2Vec on the Yelp dataset.
- G is an RNN with LSTM, pre-trained on real Yelp sequences.
- Then begin adversarial training with loss:
  \[ J(\theta) = -\mathbb{E}[R(y, y_s)] = -\sum_{s \in \mathcal{S}} G(y_n) \cdot Q(y_n) \]

**Policy Gradient**
- Given the discrete, non-differentiable input space, we use policy gradient methods to estimate updates to the policy, G:
  \[ \nabla \log Q(y|s) = \nabla \log G(y|s) \cdot \nabla \log \mathbb{P}(s) \]
- Using the REINFORCE algorithm,
  \[ \nabla J(\theta) = \sum_{s \in \mathcal{S}} \nabla \log Q(y|s) \cdot \nabla \log G(y|s) \cdot Q(y|s) \]

**Discriminator**
- D is a CNN with parallel filters of sizes [1, 2, 3, 4, 5, 8, 10, 15, 20] to emulate looking at unigram, bigram, etc.
- Pre-trained using samples from G and Yelp reviews.
- \[ J(\phi) = -\frac{1}{N} \sum \log(D_s(s)) - \frac{1}{N} \sum \log(1 - D_g(g_s)) \]

**Optimization**
- When generating sequences, we experiment with both Monte Carlo search and Beam search for generating our next token.
  - MCS: sample from complete distribution modeled by G
  - Beam Search: samples from top-k most likely next tokens and we decrease beam width over our adversarial training (limit is 1000)
- We then normalize D’s outputs in order to control the variance of the reward signal.

**Results**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>on1</th>
<th>on3</th>
<th>on4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0.02977</td>
<td>0.00001</td>
<td>0.00000</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.37574</td>
<td>0.17603</td>
<td>0.06561</td>
</tr>
<tr>
<td>GAN - BS</td>
<td>0.26772</td>
<td>0.11074</td>
<td>0.03579</td>
</tr>
<tr>
<td>GAN - MCS</td>
<td>0.29007</td>
<td>0.13661</td>
<td>0.04561</td>
</tr>
<tr>
<td>GAN - NORM</td>
<td>0.30363</td>
<td>0.16366</td>
<td>0.05618</td>
</tr>
</tbody>
</table>

**Discussion**
- We faced challenges with tuning hyperparameters as there was high instability.
- Applying GANs to real data with inconsistent content proved difficult, as opposed to modeling synthetic data as other papers have done.
- Reviews are about a mix of restaurants from a mix of people—not one theme like generating Shakespeare or Obama speeches.
- Normalizing rewards has a stabilizing effect on generator.

**Future Work**
- Incorporate new evaluation methods on quality of sentences (crowdsourcing, etc).
- Experiment and tune loss functions to increase GAN stability.
- Conditional Generation to generate different kinds of reviews based on different start tokens.

**References**