Decoding from Neural Text Generation Models

Antoine Bosselut

Stanford University
Stanford NLP
EPFL
1. At each time step, model computes a score $o_n$ for each token in our vocabulary, $w_n \in V$

$$O_n = f(\{y\}_t)$$

$f(.)$ is your model.
1. At each time step, model computes a score $o_n$ for each token in our vocabulary, $w_n \in V$

$$O_n = f(\{y\}_t)$$

2. Compute a probability distribution over these scores (usually softmax)

$$P(y_t = w_n \mid \{y\}_t) = \frac{e^{o_n}}{\sum_{m=1}^{M} e^{o_m}}$$

$O_n = f(\cdot)$ is your model

$P(\cdot)$ is your distribution over tokens
1. At each time step, model computes a score $o_n$ for each token in our vocabulary, $w_n \in V$

$$O_n = f(\{y\}_{<t})$$

$f(.)$ is your model

2. Compute a probability distribution over these scores (usually softmax)

$$P(y_t = w_n | \{y\}_{<t}) = \frac{e^{o_n}}{\sum_{m=1}^{M} e^{o_m}}$$

$P(.)$ is your distribution over tokens

3. Define a function to select a token from this distribution

$$\hat{y}_t = g(P(y_t | \{y\}_{<t}))$$

g(.) is your decoding algorithm
Simplest approach: Argmax Decoding

- \( g \) = select the token with the highest probability:

\[
\hat{y}_t = \underset{w \in V}{\text{argmax}} P(y_t = w \mid \{y\}_{<t})
\]
Simplest approach: Argmax Decoding

- \( g \) = select the token with the highest probability:

\[
\hat{y}_t = \arg\max_{w \in V} P(y_t = w \mid \{y\}_{<t})
\]

Select highest scoring token

He wanted to go to the

Model

- restroom
- grocery
- store
- airport
- pub
- gym
- bathroom
- game
- beach
- hospital
- doctor
Maybe we need more options: Beam Search

- $g = \text{cache } b \text{ paths for two steps}$
Maybe we need more options: Beam Search

- $g = \text{cache } b \text{ paths for two steps}$

He wanted to go to the

If $b = 2$, select top two tokens:

- restroom
- grocery
- store
- airport
- pub
- gym
- bathroom
- game
- beach
- hospital
- doctor
Maybe we need more options: Beam Search

- $g = \text{cache } b \text{ paths for two steps}$

Use them both as inputs to the decoder at next step

He wanted to go to the

- restroom $ightarrow$ Model
- grocery $ightarrow$ Model
Maybe we need more options: Beam Search

- \( g \) = cache 2 paths for two steps

He wanted to go to the restroom at the grocery store and shop with at the Model. Yield separate distribution for both input tokens.
Maybe we need more options: Beam Search

• $g = \text{cache } b \text{ paths for two steps}$

Select top $b$ sequence continuations across both distributions

He wanted to go to the

Model

restroom + ,

and

grocery + ,

and

store
Maybe we need more options: Beam Search

- $g = \text{cache } b \text{ paths for two steps}$

He wanted to go to the restroom and

He wanted to go to the grocery store

Repeat!
Does this penalize longer sequences?

$$s(Y) = \sum_{t=1}^{T} \log P(y_t \mid \{y\}_{<t})$$

**Shorter sequences will score better!**
Does this penalize longer sequences?

- **Solution:** Normalize by token length of sequence

\[
s(Y) = \frac{1}{|Y|} \sum_{t=1}^{|Y|} \log P(y_t | \{y\}_{<t})
\]

- **Solution:** Normalize by token length relative to reference sequence

\[
s(Y) = \frac{1}{lp(Y)} \sum_{t=1}^{|Y|} \log P(y_t | \{y\}_{<t}) \quad lp(Y) = \frac{(5 + |Y|)^\alpha}{(5 + 1)^\alpha}
\]

Wu et. al., 2016
Beam search gets repetitive and repetitive

Context: In a shocking finding, scientists discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Continuation: The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the Universidad Nacional Autónoma de México (UNAM) and the Universidad Nacional Autónoma de México (UNAM).
Beam search gets repetitive and repetitive

Context: In a shocking finding, scientists discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Continuation: Repetition is a big problem in text generation!

Holtzman et. al., ICLR 2020
Why does this happen?

I don't know.
Artifact of Maximum Likelihood Training

I don’t know.

Negative Log-likelihood

Time step

openai   lstm

Negative Log-likelihood

Time step

openai   lstm
Artifact of Maximum Likelihood Training

Negative loglikelihood decreases over time!
Beam search gets repetitive and repetitive

Worse for transformer LMs

Holtzman et. al., ICLR 2020
Beam search gets repetitive and repetitive

I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired. I'm tired.

Longer it goes, the worse it gets.

Holtzman et. al., ICLR 2020
How can we reduce repetition?

- Don’t repeat n-grams (Hacky, but works!)

- Minimize additional loss term for minimizing hidden state similarity (LSTMs)

\[ \hat{y}_t = g\left(\log P(y_t | \{y\}_{<t}) - s(h_t, h_{t-m})\right) \]

(Celikyilmaz et al., NAACL 2018, Paulus et al., ICLR 2018)
Step-by-step Maximization

Beam Search Text is Less Surprising

Holtzman et. al., ICLR 2020
Time to get random: Sampling

- $g = \text{sample a token from the distribution of tokens}$

$$\hat{y}_t \sim P(y_t = w | \{y\}_{<t})$$

He wanted to go to the

Decoder

- restroom
- grocery
- store
- airport
- pub
- gym
- bathroom
- game
- beach
- hospital
- doctor

Randomly sample token according to distribution from $P(.)$
Whoa, too \textit{random}: Temperature Scaling

- Too much randomness: distribution has too much entropy
Whoa, too \textit{random}: Temperature Scaling

• Too much randomness: distribution has too much entropy

• **Solution:** Make the distribution more “peaky” with temperature scaling

Recall: \[ P(y_t | \{y\}_<t) = \frac{e^{o_n}}{\sum_{m=1}^{M} e^{o_m}} \]
Whoa, too random: Temperature Scaling

• Too much randomness: distribution has too much entropy

• **Solution:** Make the distribution more “peaky” with temperature scaling

Recall: $P(y_t | \{y\}_{<t}) = \frac{e^{o_n}}{\sum_{m=1}^{M} e^{o_m}}$

$$P(y_t | \{y\}_{<t}) = \frac{e^{o_n} / \tau}{\sum_{m=1}^{M} e^{o_m} / \tau}$$
Whoa, too random: Temperature Scaling

- Too much randomness: distribution has too much entropy
- Solution: Make the distribution more “peaky” with temperature scaling

\[
P(y_t | \{y\}_{<t}) = \frac{e^{o_n} / \tau}{\sum_{m=1}^{M} e^{o_m} / \tau}
\]

\(\tau > 1\) \quad \text{“flatter” distribution}

\(\tau < 1\) \quad \text{“peakier” distribution}
Maybe we need fewer options: Top-\(k\) sampling

- The entire distribution over tokens is not needed at every step
- Many token choices should have no chance of being selected

(Holtzman et. al., ACL 2018; Fan et al., ACL 2018)
Maybe we need fewer options: Top-k sampling

- The entire distribution over tokens is not needed at every step
- Many token choices should have no chance of being selected
- Only sample from the top \( k \) tokens in the distribution

\[
\hat{y}_t \sim P^* (y_t = w \mid \{y\}_{<t})
\]

He wanted to go to the restroom
grocery
store
airport
pub
gym
bathroom
beach
hospital
doctor

Randomly sample token from top \( k \) highest probability tokens in \( P(.) \)

(Holtzman et. al., ACL 2018; Fan et al., ACL 2018)
Issues with top-k sampling

Top-k can cut-off too **quickly**

Top-k can cut-off too **slowly**

(Holtzman et. al., ICLR 2020)
I don’t know how many options I need: Top-$p$ sampling

- Also known as **nucleus** sampling

- Sample from subset of vocabulary where probability mass is concentrated

\[
P_1(y_t \mid \{y\}_{<t}) \quad P_2(y_t \mid \{y\}_{<t}) \quad P_3(y_t \mid \{y\}_{<t})
\]

(Holtzman et. al., ICLR 2020)
I don’t know how many options I need: Top-$p$ sampling

- Also known as **nucleus** sampling:
- Sample from subset of vocabulary where probability mass is concentrated
- Probability mass has a dynamically changing **nucleus**

*$P^*_1(y_t | \{y\}_t < t)$

*$P^*_2(y_t | \{y\}_t < t)$

*$P^*_3(y_t | \{y\}_t < t)$

*(Holtzman et. al., ICLR 2020)*
I don’t know how many options I need: Top-\(p\) sampling

- Also known as **nucleus** sampling:
- Sample from subset of vocabulary where probability mass is concentrated
- Probability mass has a dynamically changing **nucleus**

* nuclei not to scale

(Holtzman et. al., ICLR 2020)
This all sounds a bit risky

- What if my sequence just isn’t very good?
Optimize other sequence-level scores: Re-ranking

• What if my sequence just isn’t very good?

• Sample a bunch of sequences

• Define a score to approximate the quality of your sequence.

• Simplest is to just use perplexity!
Optimize other sequence-level scores: Re-ranking

• What if my sequence just isn’t very good?

• Sample a bunch of sequences

• Define a score to approximate the quality of your sequence.

• Simplest is to just use perplexity!

• However, re-rankers can be used to score a variety of properties: style (Holtzman et al., 2018), discourse (Gabriel et al., 2019), entailment/factuality (Goyal et al., 2020), logical consistency (Lu et al., 2020), and many more…
Optimize other sequence-level scores: Re-ranking

• What if my sequence just isn’t very good?

• Sample a bunch of sequences

• Define a score to approximate the quality of your sequence.
  Simplest is to just use perplexity!

• However, re-rankers can be used to score a variety of properties: style (Holtzman et al., 2018), discourse (Gabriel et al., 2019), entailment/factuality (Goyal et al., 2020), logical consistency (Lu et al., 2020), and many more…

• Change your distribution at inference time!
kNN Language Models

- Don’t just rely on your trained model to generate a distribution over tokens
- Use knowledge of similar contexts from another corpus

<table>
<thead>
<tr>
<th>Training Contexts $C_i$</th>
<th>Targets $U_i$</th>
<th>Representations $k_i = f(c_i)$</th>
<th>Distances $d_i = d(q, k_i)$</th>
<th>Nearest $k$</th>
<th>Normalization $p(k_i) \propto \exp(-d_i)$</th>
<th>Aggregation $p_{\text{kNN}}(y) = \sum_{i=1}^{k} p(k_i)$</th>
<th>Classification $p_{\text{PLM}}(y)$</th>
<th>Interpolation $p(y) = \lambda p_{\text{kNN}}(y) + (1-\lambda) p_{\text{PLM}}(y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obama was senator for</td>
<td>Illinois</td>
<td><img src="image1" alt="Representations" /></td>
<td><img src="image2" alt="Distances" /></td>
<td>Hawaii 3</td>
<td><img src="image3" alt="Normalization" /></td>
<td><img src="image4" alt="Aggregation" /></td>
<td><img src="image5" alt="Classification" /></td>
<td><img src="image6" alt="Interpolation" /></td>
</tr>
<tr>
<td>Barack is married to</td>
<td>Michelle</td>
<td></td>
<td></td>
<td>Illinois 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obama was born in</td>
<td>Hawaii</td>
<td></td>
<td></td>
<td>Hawaii 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obama is a native of</td>
<td>Hawaii</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hawaii</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test Context $x$

- Obama’s birthplace is

Target

- ?

Representation $q = f(x)$

(Khandelwal et. al., ICLR 2020)
**kNN Language Models**

- Don’t just rely on your trained model to generate a distribution over tokens
  - Use knowledge of similar contexts from another corpus

---

<table>
<thead>
<tr>
<th>Training Contexts $C_i$</th>
<th>Targets $U_i$</th>
<th>Representations $k_i = f(c_i)$</th>
<th>Distances $d_i = d(q, k_i)$</th>
<th>Nearest $k$</th>
<th>Normalization $p(k_i) \propto \exp(-d_i)$</th>
<th>Aggregation $p_{\text{avg}}(y) = \sum_{i=1}^{k} p(k_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obama was senator for</td>
<td>Illinois</td>
<td><img src="image" alt="Representations" /></td>
<td></td>
<td>Hawaii</td>
<td>Hawaii 0.7, Illinois 0.2, Hawaii 0.1</td>
<td>Hawaii 0.8, Illinois 0.2</td>
</tr>
<tr>
<td>Barack is married to</td>
<td>Michelle</td>
<td></td>
<td></td>
<td>Illinois</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obama was born in</td>
<td>Hawaii</td>
<td></td>
<td></td>
<td>Hawaii</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obama is a native of</td>
<td>Hawaii</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Context $x$</td>
<td>Target</td>
<td>Representation $q = f(x)$</td>
<td></td>
<td></td>
<td>Classification $p_{\text{PLM}}(y)$</td>
<td>Interpolation $p(y) = \lambda p_{\text{KNN}}(y) + (1-\lambda)p_{\text{PLM}}(y)$</td>
</tr>
<tr>
<td>Obama’s birthplace is</td>
<td>?</td>
<td><img src="image" alt="Representation" /></td>
<td></td>
<td></td>
<td><img src="image" alt="Classification" /></td>
<td><img src="image" alt="Interpolation" /></td>
</tr>
</tbody>
</table>

(Khandelwal et. al., ICLR 2020)

Initialize a **database** of contexts
kNN Language Models

- Don’t just rely on your trained model to generate a distribution over tokens
- Use knowledge of similar contexts from another corpus

Efficiently compute **distance** between each context in DB and current sequence

---

Training Contexts $C_i$

- Obama was senator for Barack
- Barack is married to Michelle
- Obama was born in Hawaii
- Obama is a native of Hawaii

Targets $U_i$

- Illinois
- Michelle
- Hawaii

Representations $k_i = f(c_i)$

- Hawaii
- Illinois
- Hawaii

Distances $d_i = d(q, k_i)$

- 4
- 100
- 5
- 3

Nearest $k$

- Hawaii: 3
- Illinois: 4
- Hawaii: 5

Normalization $p(k_i) \propto \exp(-d_i)$

- Hawaii: 0.7
- Illinois: 0.2
- Hawaii: 0.1

Aggregation $p_{kNN}(y) = \sum_{i=1}^{k} p(k_i)$

- Hawaii: 0.8
- Illinois: 0.2

Classification $p_{LM}(y)$

- Hawaii: 0.2
- Illinois: 0.2

Interpolation $p(y) = \lambda p_{kNN}(y) + (1 - \lambda)p_{LM}(y)$

- Hawaii: 0.6
- Illinois: 0.2

Test Context $x$

- Obama’s birthplace is

Target $q = f(x)$

- ?

Representation $q = f(x)$

(Khandelwal et. al., ICLR 2020)

Efficiently compute **distance** between each context in DB and current sequence.
kNN Language Models

- Don’t just rely on your trained model to generate a distribution over tokens
  - Compute **distribution** over possible **targets** from context DB sequences using **distance of history**

- Use knowledge of similar contexts from another corpus

(Khandelwal et. al., ICLR 2020)
kNN Language Models

- Don’t just rely on your trained model to generate a distribution over tokens
- Use knowledge of similar contexts from another corpus

![Diagram with text](image)

(Gulcehre et al., 2015; Venugopalan et al., EMNLP 2016; Khandelwal et. al., ICLR 2020)
Plug and Play Language Models!

- What if I want to encourage a tough to formalize behavior at inference time?

(Dathathri et al., ICLR 2020)
What if I want to encourage a tough to formalize behavior at inference time?

Define an attribute model that scores the generated sequence. Each generated token must try to increase the score given to the sequence by the attribute model.
Plug and Play Language Models!

- What if I want to encourage a tough to formalize behavior at inference time?

Backpropagate loss with respect to attribute score

| Attribute Model $p(a|x)$ |
|--------------------------|

The chicken tastes
Grad (Positive sentiment)
ok delicious
Original distribution ("ok")
Updated distribution ("delicious")

Updated Latents
Backward Pass and update latents
Forward Pass
Recompute with updated latents
Recompute
Updated distribution ("delicious")

(Dathathri et al., ICLR 2020)
Plug and Play Language Models!

- What if I want to encourage a tough to formalize behavior at inference time?

(Dathathri et al., ICLR 2020)

Update internal activations of language model based on these gradients.
Plug and Play Language Models!

- What if I want to encourage a tough to formalize behavior at inference time?

(Dathathri et al., ICLR 2020)
Ray hung a tire on a rope to make his daughter a swing. Ray ran to his daughter to make sure she was okay. Future constraints described by natural language:

\[
\mathcal{L}(X, \hat{Y}, Z) := - \sum_{n=1}^{N_Z} \log P_{LM}(z_n | X, \hat{Y}, Z_{1:n-1})
\]

(Qin et. al., EMNLP 2020)
Takeaways

• Decoding is a challenging problem in natural language generation

• Human language distribution is noisy and doesn’t reflect simple properties (i.e., maximization)

• Decoding algorithms can allow us to interject inductive biases that encourage properties of coherent NLG

• A lot more work to be done!
Training Neural Text Generation Models

Antoine Bosselut
Generation Model Basics

1. At each time step, model computes a score $o_n$ for each token in our vocabulary, $w_n \in V$

$$O_n = f(\{y\}_{<t})$$

2. Compute a probability distribution over these scores (usually softmax)

$$P(y_t = w_n | \{y\}_{<t}) = \frac{e^{o_n}}{\sum_{m=1}^{M} e^{o_m}}$$

3. Define a loss function to select a token from this distribution
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words
Maximizing Likelihood

• Trained to generate the next word given a set of preceding words

Text Generation Model
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words

Text Generation Model
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words
Maximizing Likelihood

• Trained to generate the next word given a set of preceding words

Text Generation Model

$y_0^* \quad y_1^* \quad y_2^* \quad y_3^* \quad y_{T-4}^* \quad y_{T-3}^* \quad y_{T-2}^* \quad y_{T-1}^*$

Allen
sailed
across
oceans
in
bought
a
boat
Maximizing Likelihood

- Trained to generate the next word given a set of preceding words

Text Generation Model

\[ y_T^* \]
\[ \text{boat} \]
\[ \text{bought} \]
\[ \text{in} \]
\[ \text{oceans} \]
\[ \text{across} \]
\[ \text{sailed} \]
\[ \text{Allen} \]
**Maximizing Likelihood**

- Trained to generate the next word given a set of preceding words

\[
\mathcal{L} = - \sum_{t=1}^{T} \log P(y^*_t | \{y^*_t\}_{<t})
\]
Issue #1: MLE discourages diversity

Context: In a shocking finding, scientists discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Continuation: The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the Universidad Nacional Autónoma de México (UNAM) and the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México...
Issue #1: MLE discourages diversity

Holtzman et. al., ICLR 2020
Alternatives: Unlikelihood Training

• Sequence-level Unlikelihood Training

• Given a set of undesired tokens $\mathcal{C}$, lower their likelihood in context

$$\mathcal{L}_{UL}^t = - \sum_{y_{neg} \in \mathcal{C}} \log(1 - P(y_{neg} | \{y^*\}_{<t}))$$

$$\mathcal{L}_{MLE}^t = - \log P(y^*_t | \{y^*\}_{<t})$$
Alternatives: Unlikelihood Training

• Sequence-level Unlikelihood Training

• Given a set of undesired tokens $\mathcal{C}$, lower their likelihood in context

\[
\mathcal{L}_{t}^{MLE} = - \log P(y^* | \{y^*\}_{<t}) \\
\mathcal{L}_{t}^{UL} = - \sum_{y_{neg} \in \mathcal{C}} \log (1 - P(y_{neg} | \{y^*\}_{<t}))
\]

Typical Negative Loglikelihood objective

Welleck et. al., ICLR 2020
Alternatives: Unlikelihood Training

• Sequence-level Unlikelihood Training

• Given a set of undesired tokens $\mathcal{C}$, lower their likelihood in context

\[ \mathcal{L}_{UL}^t = - \sum_{y_{neg} \in \mathcal{C}} \log(1 - P(y_{neg} | \{y^*\}_{<t})) \]

Typical Negative Loglikelihood objective

Unlikelihood objective lowers the probability of certain tokens

\[ \mathcal{L}_{MLE}^t = - \log P(y^*_t | \{y^*_\}_{<t}) \]
Alternatives: Unlikelihood Training

- Sequence-level Unlikelihood Training

- Given a set of undesired tokens $\mathcal{C}$, lower their likelihood in context

$$
\mathcal{L}_{MLE}^t = - \log P(y^*_t | \{y^*_t\}_{<t})
$$

$$
\mathcal{L}_{UL}^t = - \sum_{y_{neg} \in \mathcal{C}} \log (1 - P(y_{neg} | \{y^*_t\}_{<t}))
$$

$$
\mathcal{L}_{ULE}^t = \mathcal{L}_{MLE}^t + \alpha \mathcal{L}_{UL}^t
$$

Combine them for full unlikelihood training

Welleck et. al., ICLR 2020
Alternatives: Unlikelihood Training

• Sequence-level Unlikelihood Training

• Given a set of undesired tokens \( \mathcal{C} \), lower their likelihood in context

\[
\mathcal{L}^t_{MLE} = - \log P(y^*_t | \{y^*_\}_t) \\
\mathcal{L}^t_{UL} = - \sum_{y_{neg} \in \mathcal{C}} \log(1 - P(y_{neg} | \{y^*_\}_t)) \\
\mathcal{L}^t_{ULE} = \mathcal{L}^t_{MLE} + \alpha \mathcal{L}^t_{UL}
\]

But wait, what's \( \mathcal{C} \)?

Welleck et. al., ICLR 2020
Alternatives: Unlikelihood Training

- Sequence-level Unlikelihood Training

- Given a set of undesired tokens $C$, lower their likelihood in context

$$
\mathcal{L}_{MLE}^t = - \log P(y^*_t | \{y^*\}_{<t}) \\
\mathcal{L}_{UL}^t = - \sum_{y_{neg} \in C} \log(1 - P(y_{neg} | \{y^*\}_{<t})) \\
\mathcal{L}_{ULE}^t = \mathcal{L}_{MLE}^t + \alpha \mathcal{L}_{UL}^t \\
C = \{y^*\}_{<t}
$$

Welleck et. al., ICLR 2020
Alternatives: $F^2$ Softmax

- Avoid likelihood issues by factorizing the softmax
- Initialize $C$ frequency classes
- Distribute vocabulary into classes so that token frequency uniformly distributed across and between classes
Alternatives: $F^2$ Softmax

- Learn to select both frequency class and vocabulary token during training

\[ P(y_t = w_n \mid \{y\}_{<t}) = \frac{e^{U_{nh}}}{\sum_{m=1}^{M} e^{U_{mh}}} \]

\[ P(y_t = w_n \mid \{y\}_{<t}) = \left( \frac{e^{V_{jh}}}{\sum_{c=1}^{C} e^{V_{ch}}} \right) \left( \frac{e^{U_{nh}}}{\sum_{m=1}^{M} e^{U_{mh}}} \right) \]
Issue #2: Exposure Bias

• During training, we condition on gold context tokens that are real human-generated text

\[ \mathcal{L}_{MLE} = - \log P(y^*_t | \{y^*_k\}_{<t}) \]

• During inference, we decode from distributions conditioned on previously generated tokens

\[ \mathcal{L}_{dec} = - \log P(\hat{y}_t | \{\hat{y}_k\}_{<t}) \]
Issue #2: Exposure Bias

Holtzman et. al., ICLR 2020
Issue #2: Exposure Bias

- During training, we condition on gold context tokens that are real human-generated text

\[ \mathcal{L}_{MLE} = - \log P(y^*_t | \{y^*_t\}_{<t}) \]

- During inference, we decode from distributions conditioned on previously generated tokens

\[ \mathcal{L}_{dec} = - \log P(\hat{y}_t | \{\hat{y}_t\}_{<t}) \]
Reinforcement Learning

- Cast a text generation model as a MDP
  - State is denoted by a preceding context
  - Actions are the words you can generate
  - Policy is the text generation model
  - Rewards are provided by an external source
REINFORCE

- Trained to generate the next word given a set of preceding words

$$\mathcal{L}_{RL} = -\sum_{t=T-3}^{T} r(\hat{y}_t) \log P(\hat{y}_t | \{\hat{y}\}_t; \{y^*\})$$

Text Generation Model

```
    START
  y0  y1  y2  y3
  \hat{y}_{T-3}  \hat{y}_{T-2}  \hat{y}_{T-1}  \hat{y}_T
```
Reward Estimation

• How do we define a reward function?
  • BLEU (machine translation; Ranzato et al., ICLR 2016; Wu et al., 2016)
  • ROUGE (summarization; Paulus et al., ICLR 2018; Celikyilmaz et al., NAACL 2018)
  • CIDEr (image captioning; Rennie et al., CVPR 2017)
  • SPIDEr (image captioning; Liu et al., ICCV 2017)
Reward Estimation

• How do we define a reward function?

  • BLEU (machine translation; Ranzato et al., ICLR 2016; Wu et al., 2016)

  • ROUGE (summarization; Paulus et al., ICLR 2018; Celikyilmaz et al., NAACL 2020)

  • CIDEr (image captioning; Rennie et al., CVPR 2017)

  • SPIDEr (image captioning; Liu et al., ICCV 2017)

Optimizing for the task vs. Gaming the reward
What behaviors can we tie to rewards?

- Cross-modal consistency (Ren et al., CVPR 2017)
- Simplicity (Zhang and Lapata, EMNLP 2017)
- Temporal consistency (Bosselut et al., NAACL 2018)
- Politeness (Tan et al., TACL 2018)
- Paraphrasing (Li et al., EMNLP 2018)
- Sentiment (Gong et al., NAACL 2019)
- Formality (Gong et al., NAACL 2019)
Implementation Thoughts

• Credit Assignment

\[ r(\hat{y}_t) \quad \text{vs.} \quad r(\hat{Y}) \]
Implementation Thoughts

- Credit Assignment
  \[ r(\hat{y}_t) \text{ vs. } r(\hat{Y}) \]

- Set appropriate baseline
  \[ \mathcal{L}_{RL} = - \sum (r(\hat{y}_t) - b) \log P(\hat{y}_t | \{\hat{y}\}_{<t}; \{y^*\}) \]
Implementation Thoughts

• Credit Assignment

\[ r(\hat{y}_t) \text{ vs. } r(\hat{Y}) \]

• Set appropriate baseline

\[ \mathcal{L}_{RL} = - \sum (r(\hat{y}_t) - b) \log P(\hat{y}_t | \{\hat{y}\}_{<t}; \{y^*\}) \]

• Mix with MLE

\[ \mathcal{L} = \mathcal{L}_{MLE} + \alpha \mathcal{L}_{RL} \]
What if you don’t know what to use as reward?

• Adversarial Learning!

• Use an adversarially-learned scoring function to provide rewards

• Still often uses REINFORCE

• Dialogue systems (Li et al., EMNLP 2017), Visual storytelling (Wang et al., ACL 2018)
Human-in-the-loop Learning

1. **Collect human feedback**
   - A Reddit post is sampled from the Reddit TL;DR dataset.
   - Various policies are used to sample a set of summaries.
   - Two summaries are selected for evaluation.
   - A human judges which is a better summary of the post.
   - “j is better than k”

2. **Train reward model**
   - One post with two summaries judged by a human are fed to the reward model.
   - The reward model calculates a reward $r$ for each summary.
   - The loss is calculated based on the rewards and human label, and is used to update the reward model.
   - $\text{loss} = \log(\sigma(r - r_j))$
   - “j is better than k”

3. **Train policy with PPO**
   - A new post is sampled from the dataset.
   - The policy $\pi$ generates a summary for the post.
   - The reward model calculates a reward for the summary.
   - The reward is used to update the policy via PPO.

(Stiennon et al., NeurIPS 2020)
Takeaways

• Maximum likelihood estimation is still the premier algorithm for training text generation models

• Diversity is an issue with MLE, so new approaches focus on mitigating the effects of common words

• Exposure bias causes text generation models to lose coherence easily, so learning from its own samples is a promising way forward

• Reinforcement learning allows models to learn tough to quantify behaviors

• Much more!
Decoding References

[3] Venugopalan et al., Improving LSTM-based Video Description with Linguistic Knowledge Mined from Text. EMNLP 2016
[4] Li et al., A Diversity-Promoting Objective Function for Neural Conversation Models. EMNLP 2018
[7] Holtzman et al., Learning to Write with Cooperative Discriminators. ACL 2018
[8] Fan et al., Hierarchical Neural Story Generation. ACL 2018
Training References

[6] Liu et al., Improved Image Captioning via Policy Gradient Optimization of SPIDER. ICCV 2017
[8] Li et al., Adversarial Learning for Neural Dialogue Generation. EMNLP 2017
[9] Paulus et al., A Deep Reinforced Model for Abstractive Summarization. ICLR 2018
[10] Celikyilmaz et al., Deep Communicating Agents for Abstractive Summarization. NAACL 2018
[14] Li et al., Paraphrase Generation with Deep Reinforcement Learning. EMNLP 2018
[15] Gong et al., Reinforcement Learning Based Text Style Transfer without Parallel Training Corpus. NAACL 2019
[16] Holtzman et. al., The Curious Case of Neural Text Degeneration. ICLR 2020
[18] Stiennon et al., Learning to summarize from human feedback. NeurIPS 2020