Oversmoothing in neural autoregressive modeling

Kyunghyun Cho
Work done together with Ilia Kulikov and Maksim Eremeev

Language modeling

- Setup:
  - $Y = (y_1, y_2, \ldots, y_L)$, where $y_l \in V$ with V a set of all unique tokens.
  - The length $L \in \{1,2,3,\ldots\}$ may vary from one sequence to another.
  - V includes a special symbol $\langle \text{eos} \rangle$, and $y_L = \langle \text{eos} \rangle$ for any Y.
- Goal:
  - Build a model (parametrized by θ) that computes a probability $p(Y; \theta)$.
  - A reasonable Y receives a high probaiblity

Language modeling is fascinating

because language is a fascinating way to encapsulate ideas

- Relation extraction
  - Barak Obama **was the 44th president of** the United States.
- Common-sense reasoning
  - Jane performed an emergency surgery on John, because John was in a **car accident**.
- Question-answering
  - Wikipedia is **a free content, multilingual online encyclopedia written and maintained by a community of volunteers through a model of open collaboration, using a wiki-based editing system.**
- Machine translation
  - “뉴욕대학교는 뉴욕시에 위치한 사립대학교이다.” is in English “**New York University is a private university located in New York City.**”

Language modeling is fascinating

because language is a fascinating way to encapsulate ideas

- All these problems can be solved by solving

$$\arg\max_{y_{\text{missing}}} \log p([y_{\text{observed}}, y_{\text{missing}}]; \theta)$$

- So, all we need is a language model.

Autoregressive language modeling

- (Closely related) properties of language modeling
  - We deal with variable-length sequences.
  - Each element within a sequence is from a finite set of unique tokens.

Autoregressive language modeling

- Autoregressive modeling

$$p(Y;\theta) = \prod_{t=1}^{L} p(y_t \mid y_{1:t-1};\theta)$$

- One classifier is used over and over
  - It can handle sequences of varying length.
  - It is compositional (rather simple one, though.)

Autoregressive language modeling

- Autoregressive modeling

$$p(Y;\theta) = \prod_{t=1}^{L} p(y_t \mid y_{1:t-1};\theta)$$

- A natural choice for language modeling since 1950

2. ENTROPY CALCULATION FROM THE STATISTICS OF ENGLISH

One method of calculating the entropy // is by a series of approximations 7? $F_1, F_2 > ' ' '$, which successively take more and more of the statistics f the language into account and approach H as a limit. F_N may be called the N-gram entropy; it measures the amount of information or entropy due to statistics extending over N adjacent letters of text. F_N is given by[1]

$$FN = -\sum_{i,j} p(b_i, j) \log_2 p_{b_i}(j)$$
$$= -\sum_{i,j} p(b_i, j) \log_2 p(b_i, j) + \sum_i p(b_i) \log p(b_i) \quad (1)$$

in which: bi is a block of N-1 letters [(N-1)-gram]

j is an arbitrary letter following b_i

$p(b_i, j)$ is the probability of the N-gram b_i, j

$p_{b_i}(j)$ is the conditional probability of letter j after the block b_i, and is given by $p(b_i, j)/p(b_i)$.

The equation (1) can be interpreted as measuring the average uncertainty (conditional entropy) of the next letter /' when the preceding N-1 letters are known. As N is increased, F_N includes longer and longer range statistics and the entropy, H, is given by the limiting value of F_N as $N \to \infty$:

$$H = \lim_{N\to\infty} F_N . \quad (2)$$

The N-gram entropies F_N for small values of N can be calculated from standard tables of letter, digram and trigram frequencies.[2] If spaces and punctuation are ignored we have a twenty-six letter alphabet and F_0 may be taken (by definition) to be $\log_2 26$, or 4.7 bits per letter. F_1 involves letter frequencies and is given by

$$F_1 = -\sum_{i=1}^{26} p(i) \log_2 p(i) = 4.14 \text{ bits per letter.} \quad (3)$$

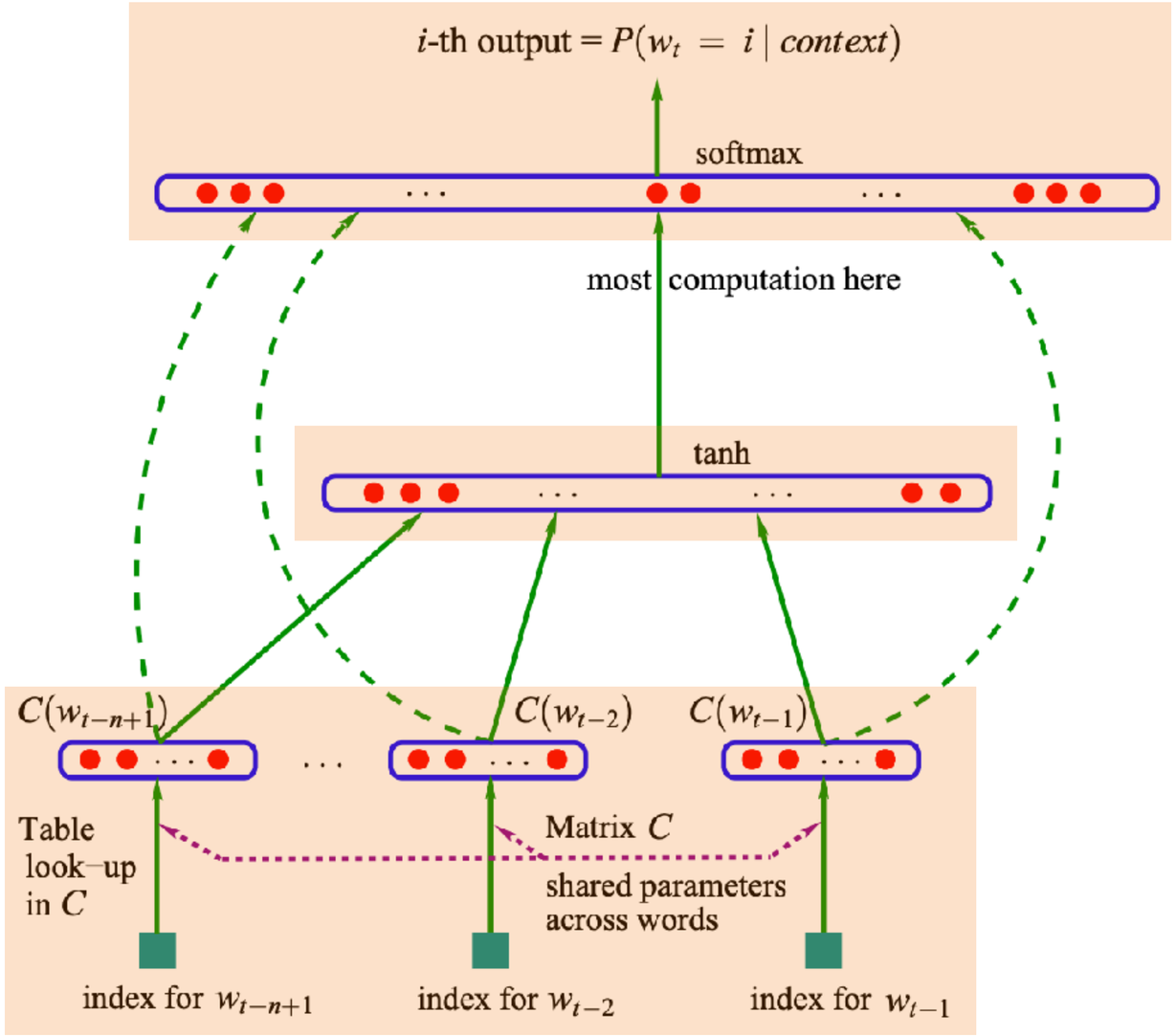
[Shannon, 1950]

Autoregressive language modeling

- Autoregressive modeling

$$p(Y;\theta) = \prod_{t=1}^{L} p(y_t \mid y_{1:t-1};\theta)$$

- A natural choice for language modeling since 1950.
- Extended to a neural version in 2001.



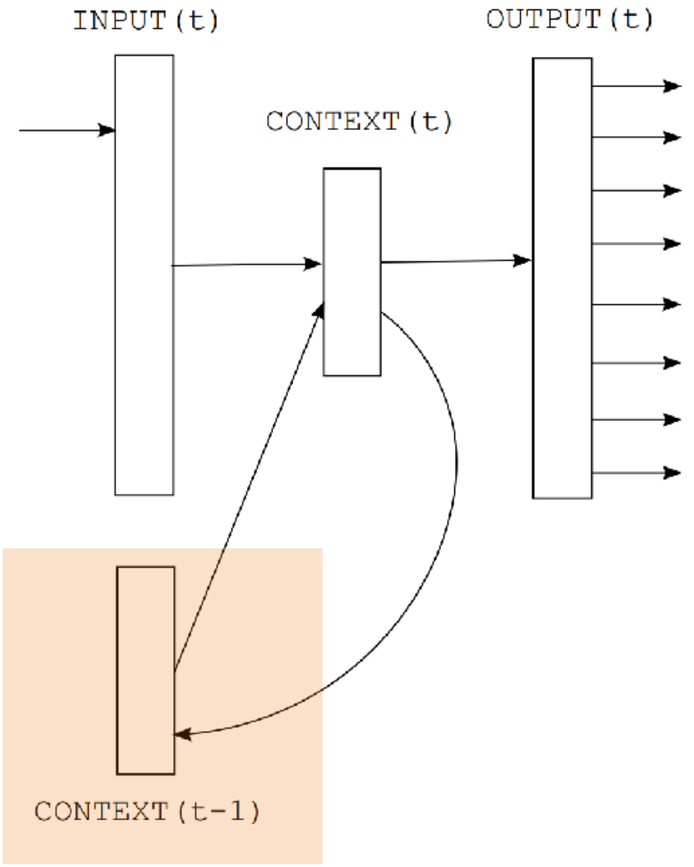
[Bengio et al., 2003]

Autoregressive language modeling

- Autoregressive modeling

$$p(Y;\theta) = \prod_{t=1}^{L} p(y_t \mid y_{1:t-1};\theta)$$

- A natural choice for language modeling since 1950.
- Extended to a neural version in 2001.
- Infinite-context version in 2011:
  - This is when things got so much more interesting.



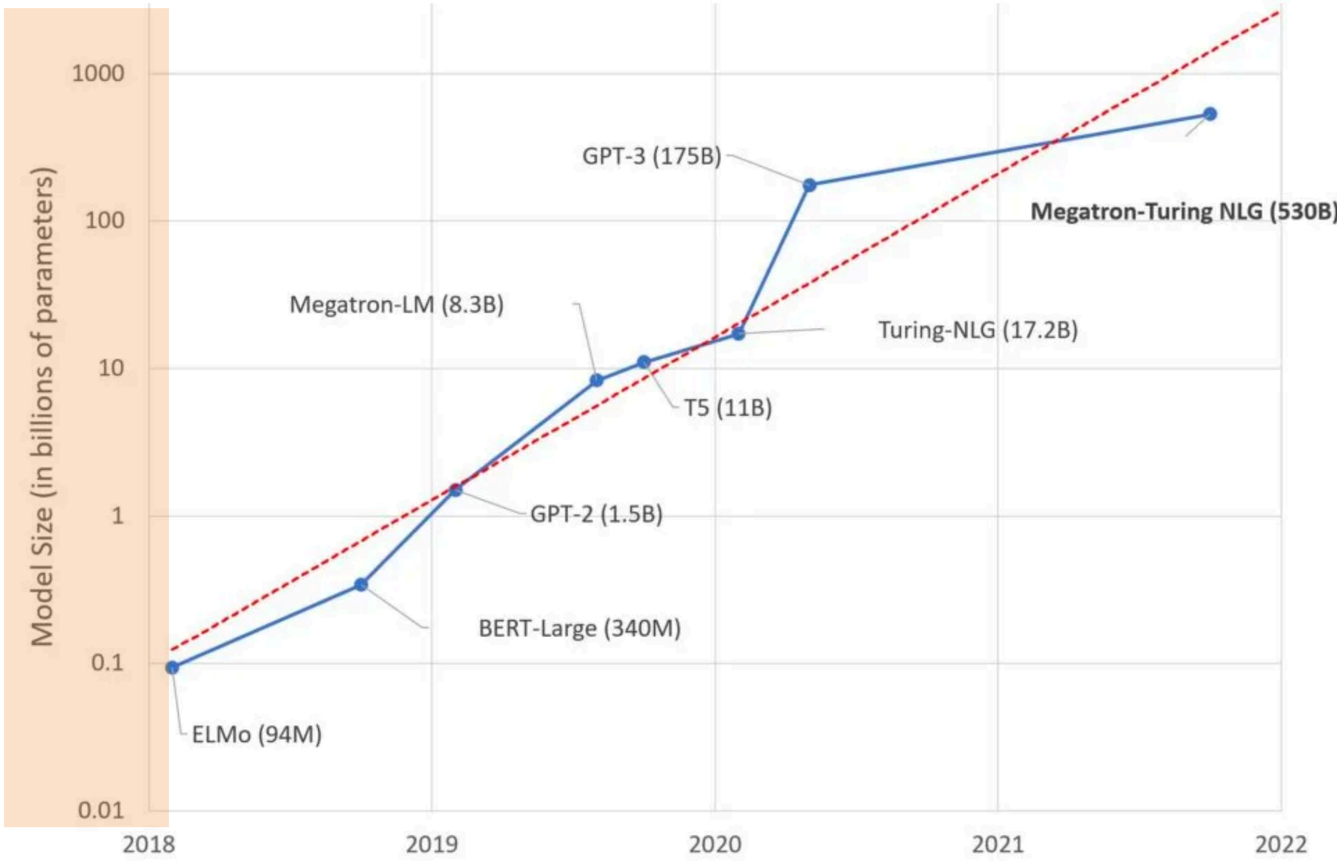
[Mikolov et al., 2011]

Autoregressive language modeling

- Autoregressive modeling

$$p(Y;\theta) = \prod_{t=1}^{L} p(y_t \mid y_{1:t-1};\theta)$$

- A natural choice for language modeling since 1950.
- A neural version in 2001.
- Infinite-context version in 2011.
- Now: the war of “# parameters”



Microsoft

Already outdated

This is a bit ridiculous that we are spending so many scientists on this war, though…

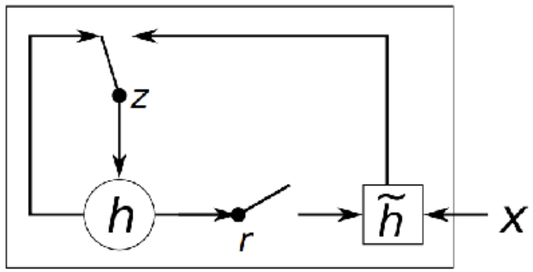
Neural autoregressive language modeling

- Language modeling is reduced to text classification: $p(y_t \mid y_1, \ldots, y_{t-1}; \theta)$
  - Target $y_t \in V$
  - Input $(y_1, \ldots, y_{t-1}) \in \underbrace{V \times \cdots \times V}_{t-1}$
  - Model parameters θ

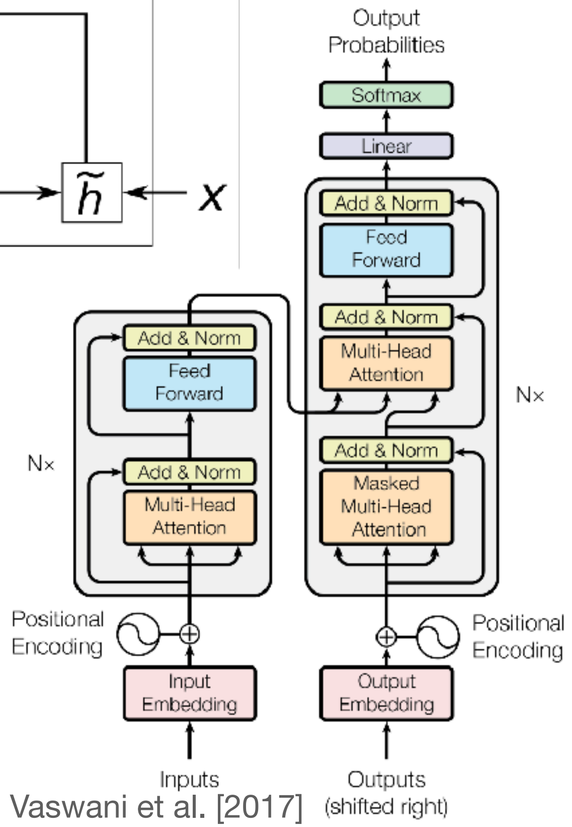
Autoregressive language modeling

- Two natural ways to build this classifier $p(y_t \mid y_1, \ldots, y_{t-1}; \theta)$
  - A recurrent network: LSTM, GRU, etc.
    - **Pro**s: constant memory/computation in the forward pass
    - **Con**s: difficult learning and inherent sequential processing
  - An attention network: transformers, etc.
    - **Pro**s: highly parallel processing and easy learning
    - **Con**s: linear memory/computation in the forward pass

But, this is not what I want to talk about it today…



Cho et al. [2014]



Vaswani et al. [2017]

Training a language model

- Language modeling is reduced to text classification: $p(y_t \mid y_1, \ldots, y_{t-1}; \theta)$
- For learning, we use cross-entropy:
  - Entropy: expected negative log-probability under the same distribution

$$FN = -\sum_{V} p(b_i, j) \log_2 p_{b_i}(j)$$

[Shannon, 1950]

  - Cross-entropy: expected negative log-probability under a different distribution

$$-\sum_{y \in V} p^*(y \mid x) \log p(y \mid x; \theta)$$

Training a language model

almost exactly like training a classifier

- For training a language model, we use cross-entropy which is the upperbound to entropy.
- In other words, training a language model is find a distribution that matches the inherent entropy of natural language.

$$-\sum_{y \in V} p^*(y|x) \log p(y|x;\theta) \geq -\sum_{y \in V} p^*(y|x) \log p^*(y|x),$$

because

$$\begin{aligned}
\sum_{y \in V} p^*(y|x) \log p(y|x;\theta) &= \sum_{y \in V} p^*(y|x) \log \frac{p^*(y|x)}{p^*(y|x)} p(y|x;\theta) \\
&= \sum_{y \in V} p^*(y|x) \left(\log p^*(y|x) + \log \frac{p(y|x;\theta)}{p^*(y|x)} \right) \\
&= \sum_{y \in V} p^*(y|x) \log p^*(y|x) + \underbrace{\sum_{y \in V} p^*(y|x) \log \frac{p(y|x;\theta)}{p^*(y|x)}}_{= -\mathrm{KL}(p^* \| p^\theta) \leq 0} \\
&\leq \sum_{y \in V} p^*(y|x) \log p^*(y|x).
\end{aligned}$$

Training a language model

Almost exactly like training a classifier

- Overall, the loss function is

$$\mathbb{E}_{Y\sim D}\left[\sum_{t=1}^{|Y|}\log p(y_t \mid y_{<t};\theta)\right] \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{|Y^n|}\log p(y_t^n \mid y_{<t}^n;\theta)$$

- **Almost** like training a classifier, but not quite.
- Examples $(y_{<t}^n, y_t^n)$ are highly correlated within each sequence Y^n.
  - Highly correlated, because they come from the same text.

But, it turned out no one really cares about this… ugh…

Taking a step back …

- Setup:
  - $Y = (y_1, y_2, \ldots, y_L)$, where $y_l \in V$ with V a set of all unique tokens.
  - The length $L \in \{1,2,3,\ldots\}$ may vary from one sequence to another.
  - V includes a special symbol $\langle \text{eos} \rangle$, and $y_L = \langle \text{eos} \rangle$ for any Y.
- Goal:
  - Build a model (parametrized by θ) that computes a probability $p(Y; \theta)$.
  - A reasonable Y receives a high probaiblity

If <eos> wasn't there …

- A prefix $y_{1:t}$ is always more likely than the full sequence $y_{1:T}$:

$$p(y_1)p(y_2|y_1)\cdots p(y_t|y_{<t}) \geq p(y_1)p(y_2|y_1)\cdots p(y_t|y_{<t})\cdots p(y_T|y_{<T})$$

- because the probability is bounded from above by 1.
- Weird…

But, <eos> is here …

- <eos> saves us from this weird behaviour.
  - Any valid sequence ends with <eos>.
- A prefix $y_{1:t}$ is always more likely than the full sequence $y_{1:T}$:
  - Instead of a prefix $y_{1:t}$, consider a premature sequence $y_{1:t} + \langle \text{eos} \rangle$

$$p(y_1)p(y_2 \mid y_1) \cdots p(y_t \mid y_{<t}) p(\langle \text{eos} \rangle \mid y_{\leq t}) \; ? \; p(y_1)p(y_2 \mid y_1) \cdots p(y_t \mid y_{<t}) \cdots p(y_T \mid y_{<T})$$

But, <eos> is here …

- A prefix $y_{1:t}$ is always more likely than the full sequence $y_{1:T}$:
  - Instead of a prefix $y_{1:t}$, consider a premature sequence $y_{1:t}+\langle\text{eos}\rangle$

$$p(y_1)p(y_2\,|\,y_1)\cdots p(y_t\,|\,y_{<t})p(\langle\text{eos}\rangle\,|\,y_{\leq t}) \;?\; p(y_1)p(y_2\,|\,y_1)\cdots p(y_t\,|\,y_{<t})\cdots p(y_T\,|\,y_{<T})$$

- The probability of a premature seq. should be lower than the full sequence:

$$\cancel{p(y_1)p(y_2\,|\,y_1)\cdots p(y_t\,|\,y_{<t})}p(\langle\text{eos}\rangle\,|\,y_{\leq t}) \leq \cancel{p(y_1)p(y_2\,|\,y_1)\cdots p(y_t\,|\,y_{<t})}p(y_{t+1}\,|\,y_{<t+1})\cdots p(y_t\,|\,y_{<t})\cdots p(y_T\,|\,y_{<T})$$

- In other words, as long as <eos> is *highly* unlikely after a prefix, the premature sequence can be less probable than the full sequence.

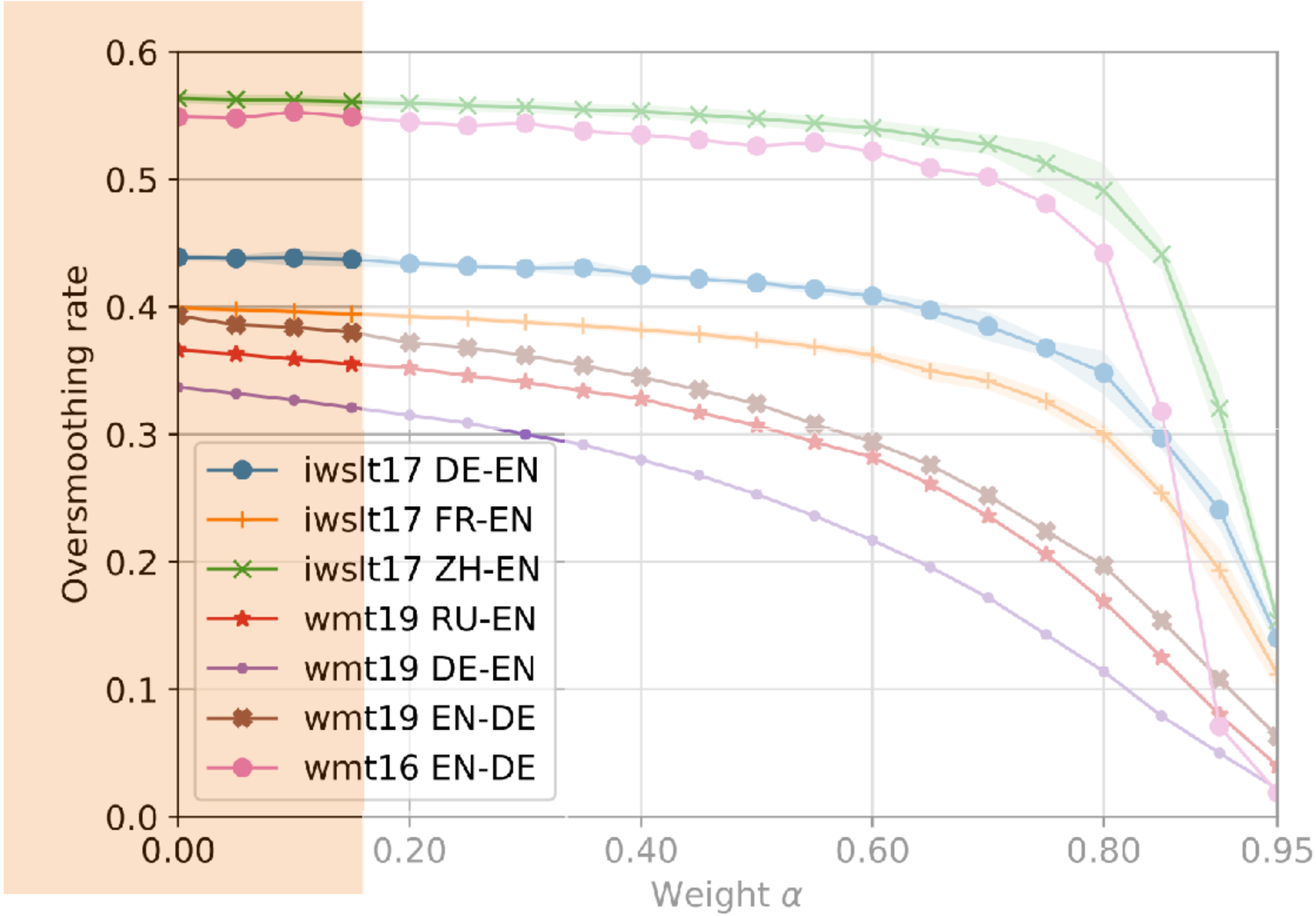
Oversmoothing rate

- Given a sequence $y = (y_1, y_2, \ldots, y_T)$, we check how often the inequality is violated:

$$r(y) = \frac{1}{T} \sum_{t=1}^{T} I\left(p(\langle \text{eos} \rangle \,|\, y_{<t}) > \prod_{t'=t}^{T} p(y_{t'} \,|\, y_{<t'}) \right)$$

- It counts how many premature seq.'s are more probable than the full seq.
- We want this oversmoothing rate to be small (≈ 0) in natural language.

Oversmoothing rate is high!



Oversmoothing rate is high!

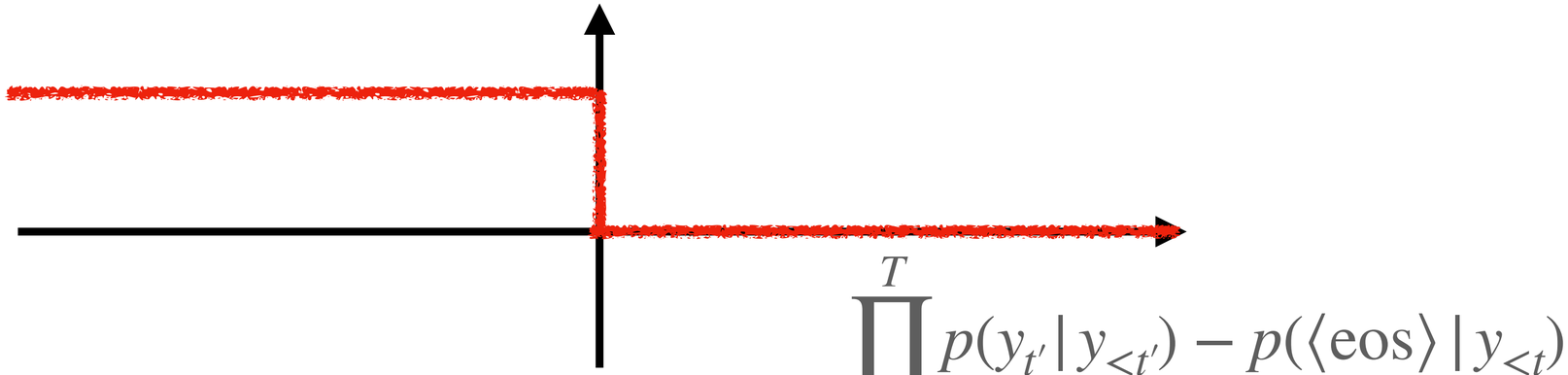
- It is not enough to simply lower $p(\langle \text{eos} \rangle \mid y_{<t})$ when $y_t \neq \langle \text{eos} \rangle$.
  - i.e., it is not enough to treat it as a series of independent classification.
- $p(\langle \text{eos} \rangle \mid y_{<t})$ must be lowered below $\prod_{t'=t}^{T} p(y_{t'} \mid y_{<t'})$.
  - i.e., we must take into account **the sequence structure**.

Minimizing the oversmoothing rate

- The oversmoothing rate is not easy to minimize

$$r(y) = \frac{1}{T} \sum_{t=1}^{T} I\left(p(\langle \text{eos} \rangle \,|\, y_{<t}) > \prod_{t'=t}^{T} p(y_{t'} \,|\, y_{<t'}) \right)$$

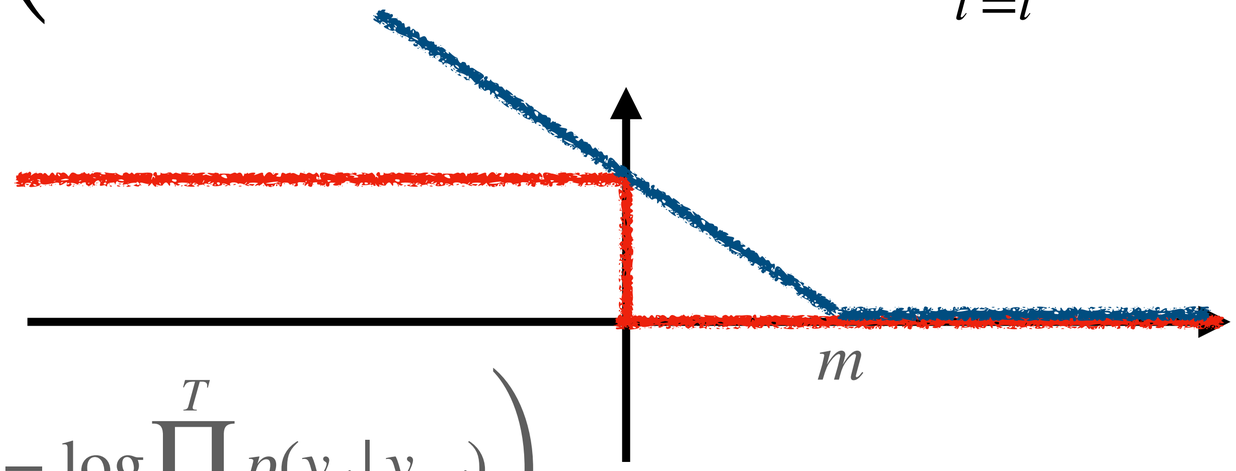
- because it is an average of 0-1 losses (piece-wise constant.)



Oversmoothing loss

- A convex relaxation with a piece-wise linear loss and margin $m \geq 0$:

$$o(y) = \frac{1}{T} \sum_{t=1}^{T} \max \left(0, m + \log p(\langle \text{eos} \rangle \,|\, y_{<t}) - \log \prod_{t'=t}^{T} p(y_{t'} \,|\, y_{<t'}) \right)$$



$$\max \left(0, m + \log p(\langle \text{eos} \rangle \,|\, y_{<t}) - \log \prod_{t'=t}^{T} p(y_{t'} \,|\, y_{<t'}) \right)$$

Oversmoothing loss

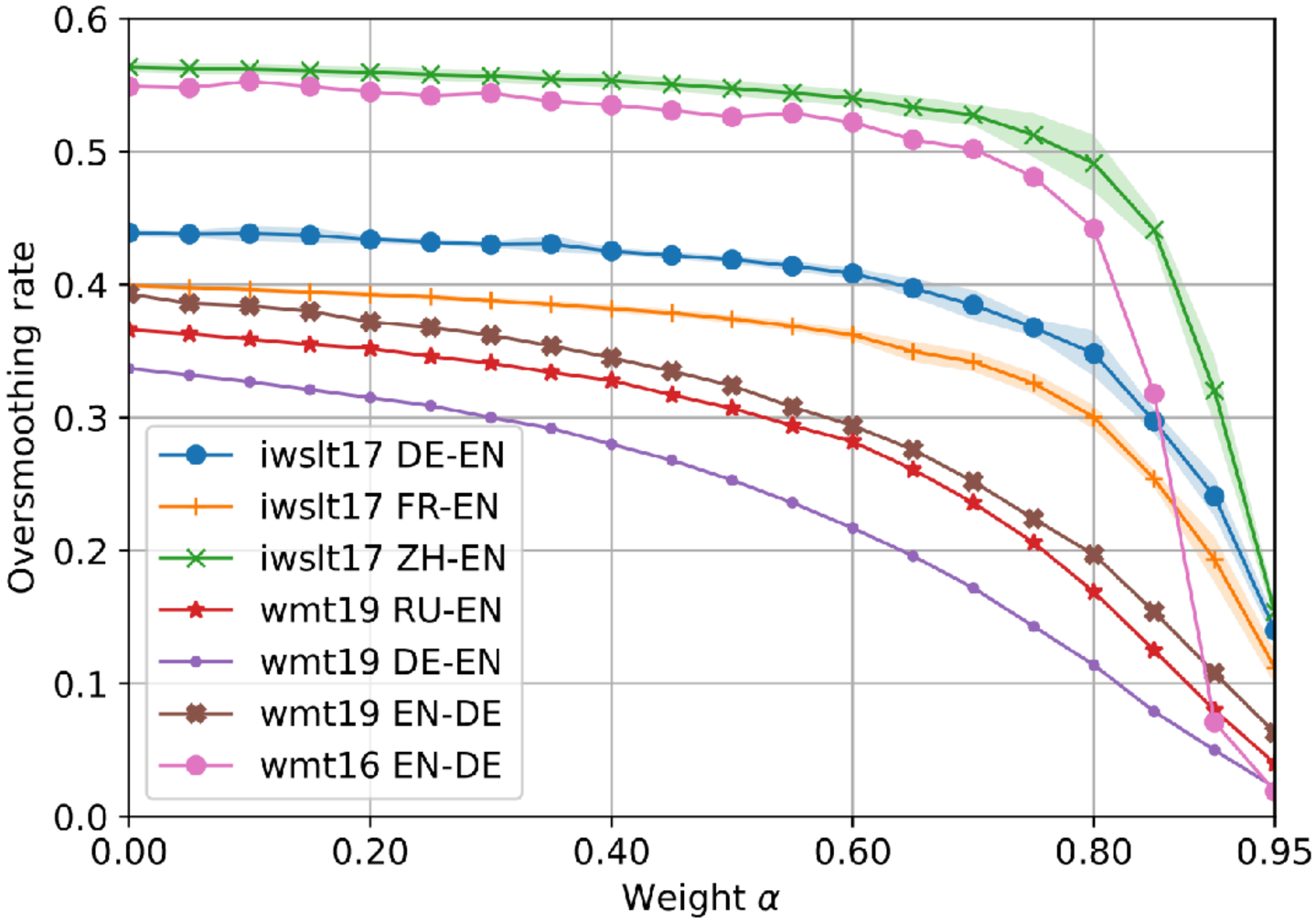
- A convex relaxation with a piece-wise linear loss and margin $m \geq 0$:

$$o(y) = \frac{1}{T} \sum_{t=1}^{T} \max \left(0, m + \log p(\langle \text{eos} \rangle \mid y_{<t}) - \sum_{t'=t}^{T} \log p(y_{t'} \mid y_{<t'}) \right)$$

- Minimizing $\log p(\langle \text{eos} \rangle \mid y_{<t})$ ensures <eos> is scored low when it is not supposed to show up: we treat <eos> specially within a sequence.
- Maximizing $\sum_{t'=t}^{T} \log p(y_{t'} \mid y_{<t'})$ ensures the model does not focus too much on the earlier tokens but also emphasize the tokens in the latter part of a sequence.

We can lower the oversmoothing rate.

Dramatically!



What happens under the hood?

- The log-probability of an incorrectly place <eos> decreases dramatically.

$$o(y) = \frac{1}{T}\sum_{t=1}^{T} \max\left(0, m + \log p(\langle \text{eos} \rangle \mid y_{<t}) - \sum_{t'=t}^{T} \log p(y_{t'} \mid y_{<t'})\right)$$

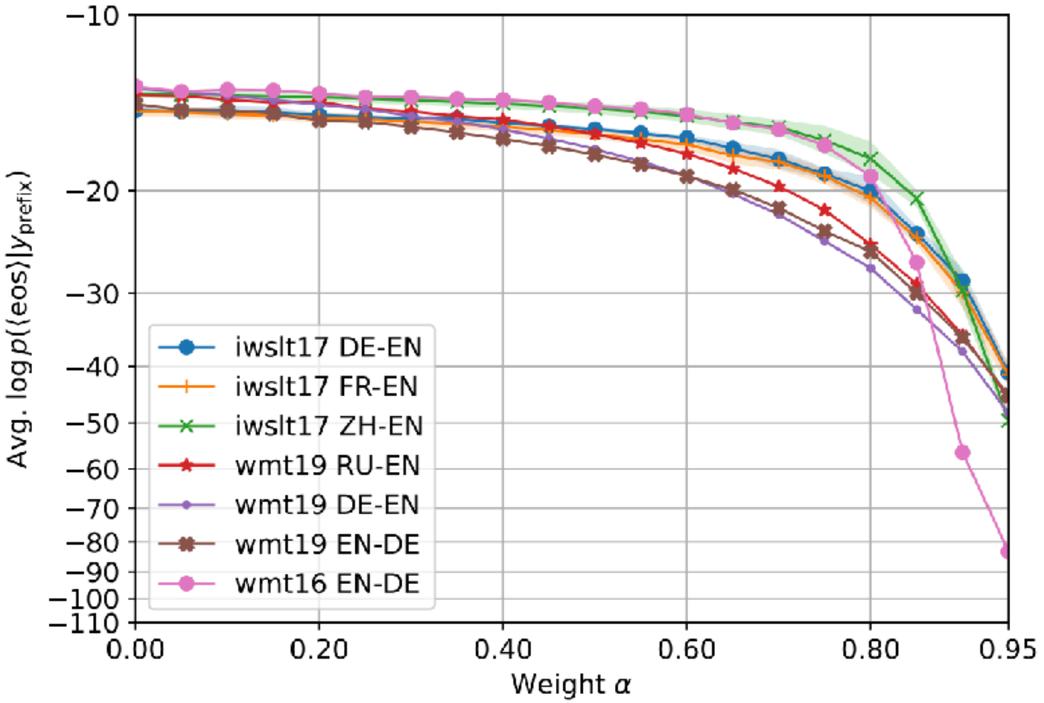


Figure 2: Log-probabilities of ⟨eos⟩ token within length-t prefixes averaged across all positions per translation and then averaged across all translations.

What happens under the hood?

- The effect is however much more visible when we look at their rank.
- The rank is what matters when we consider approximate decoding.

Theorem 3.4 (Inconsistency of an incomplete decoding algorithm). *There exists a consistent recurrent LM p_θ from which an incomplete decoding algorithm $\mathcal{F}$, that considers only up to $(|V|-1)$-most likely tokens according to $p_\theta(y_t \mid y_{<t}, C)$ at each step t, finds an infinite-length sequence $\tilde{Y}$ with probability 1, i.e., $q_{\mathcal{F}}(|Y| = \infty) = 1$.*

Welleck et al. (2020)

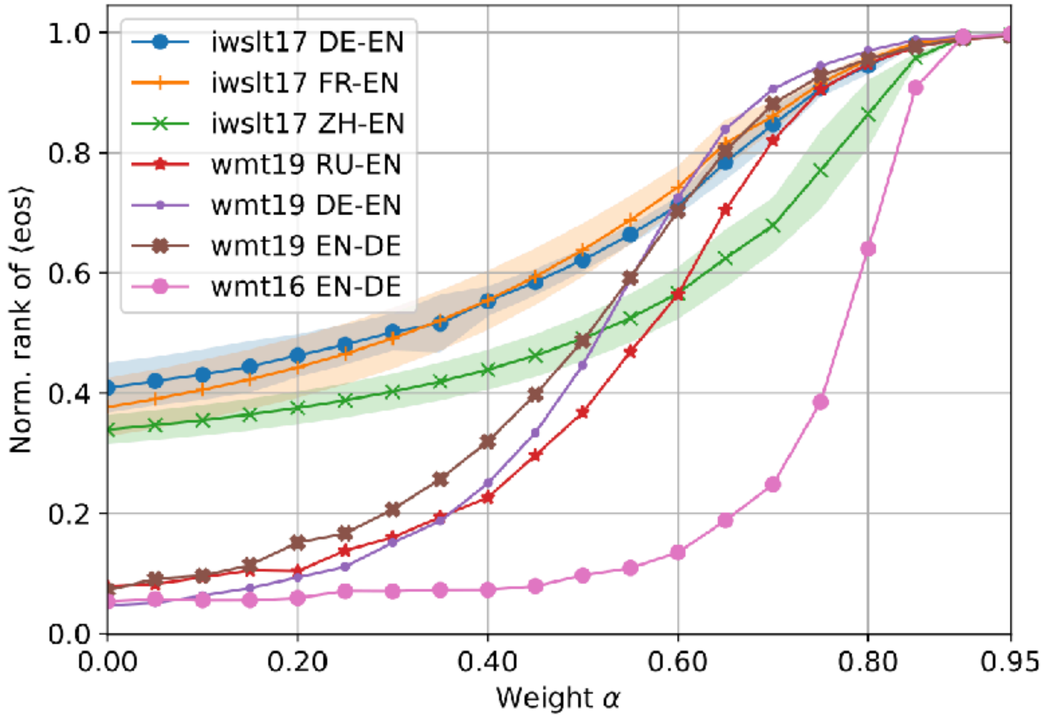


Figure 4: Normalized rank of ⟨eos⟩ token within length-t prefixes averaged across all positions per translation and then averaged across all translations.

But, it doesn’t hurt nor improve modeling

- This implies that
  - MLE has ambiguity in modeling <eos>
  - Oversmoothing loss selects a solution that avoids the issue of oversmoothing

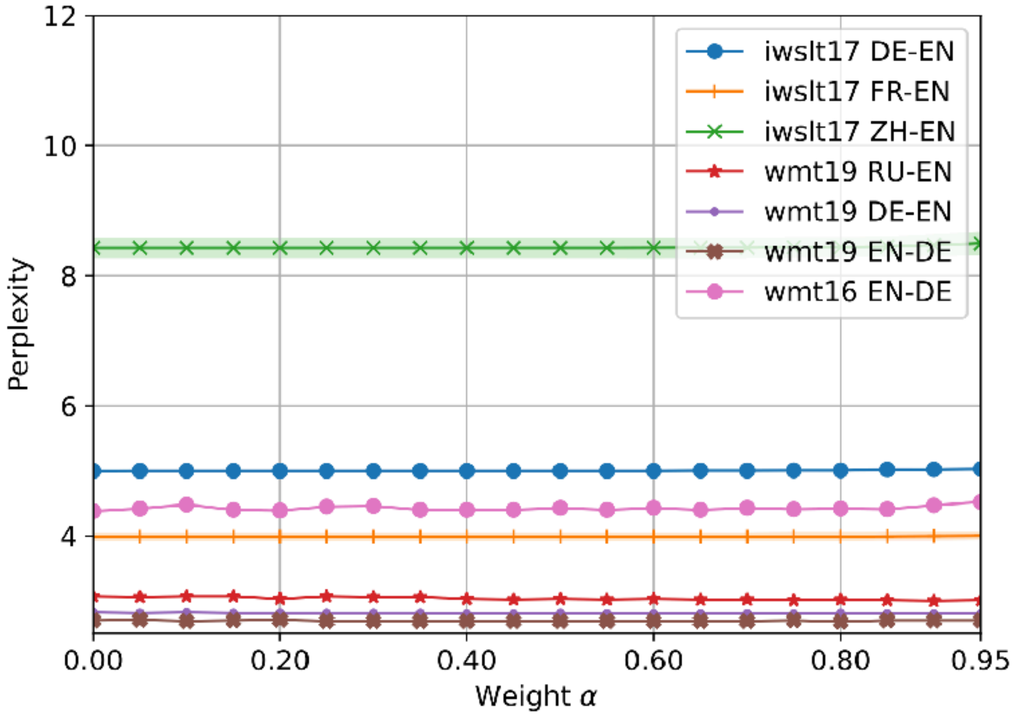


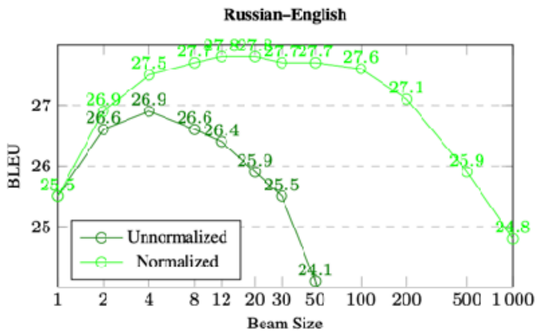
Figure 3: Perplexity measured on reference translations is stable as oversmoothing loss has more contribution.

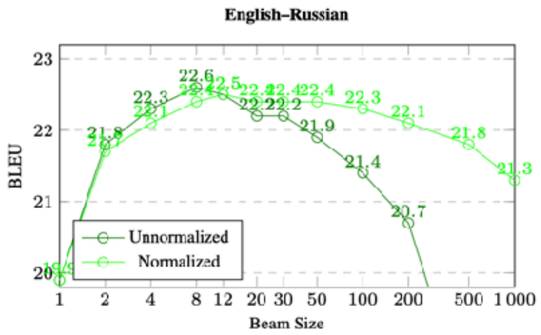
Let's take a quick step back …

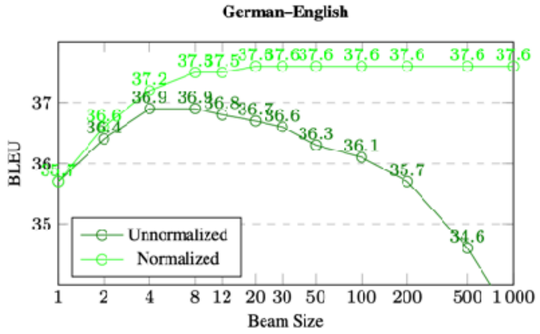
Oversmoothing in the wild

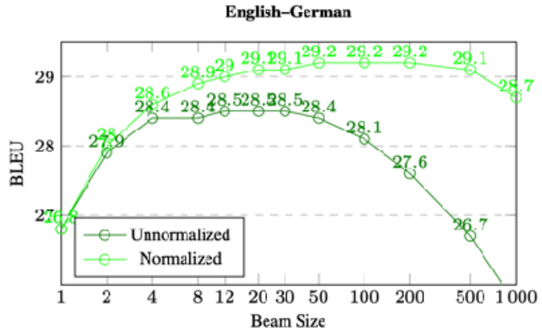
Koehn & Knowles in 2016

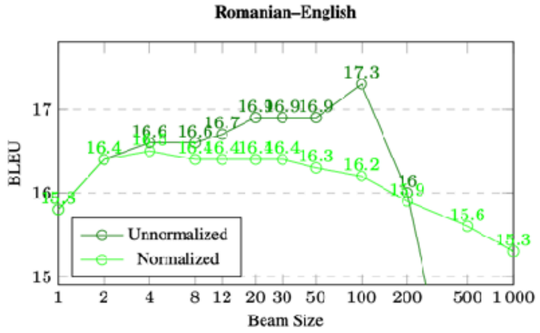
- Better search (larger beam) leads to worse translation.
- Worse translations tend to be dramatically shorter.

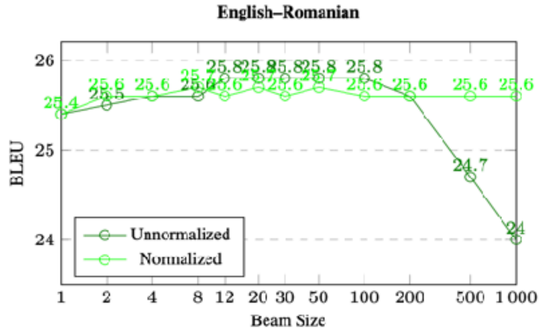


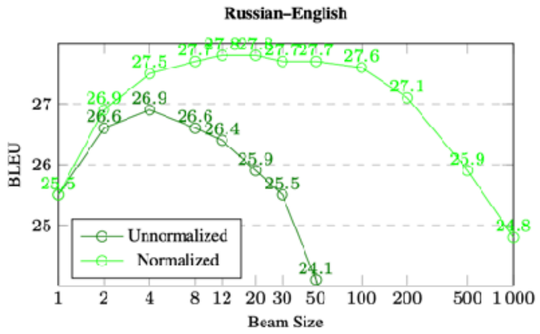


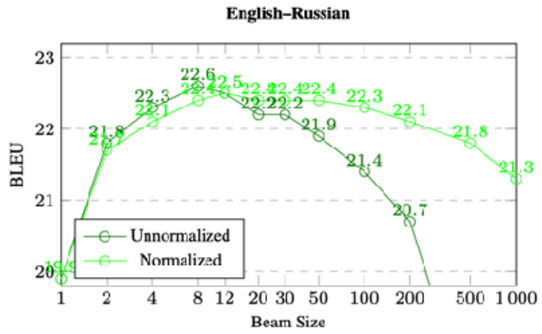












Oversmoothing in the wild

Stahlberg & Byrne in 2019

- For many source sentences, the most probable translations are *empty*!

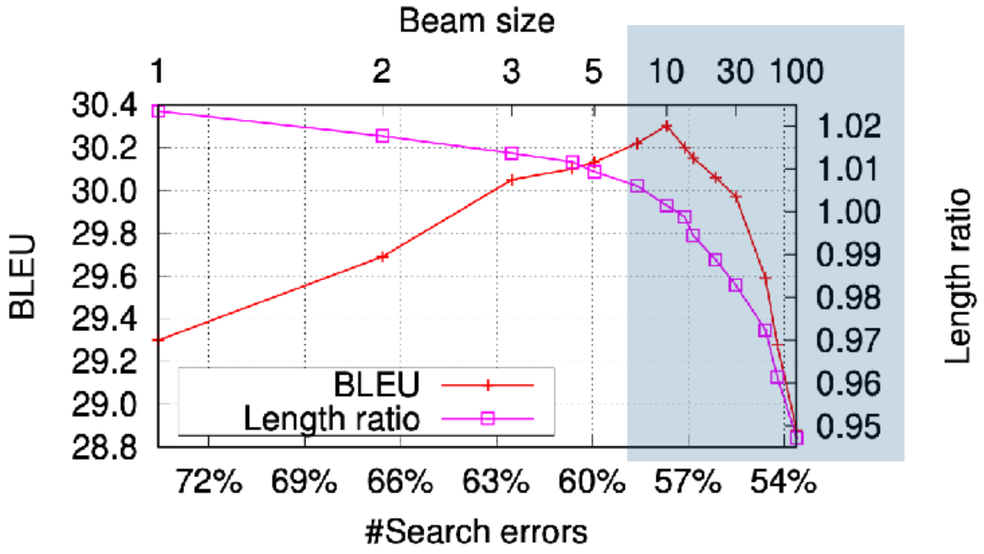


Figure 1: BLEU over the percentage of search errors. Large beam sizes yield fewer search errors but the BLEU score suffers from a length ratio below 1.

Model	Beam-10 BLEU	Beam-10 #Search err.	Exact #Empty
LSTM*	28.6	58.4%	47.7%
SliceNet*	28.8	46.0%	41.2%
Transformer-Base	30.3	57.7%	51.8%
Transformer-Big*	31.7	32.1%	25.8%

Table 2: *: The recurrent LSTM, the convolutional SliceNet (Kaiser et al., 2017), and the Transformer-Big systems are strong baselines from a WMT'18 shared task submission (Stahlberg et al., 2018a).

Oversmoothing in the wild

Post-hoc fixes

- Length penalty during decoding [Cho et al., 2014; Wu et al., 2016; …]

When we use the beam-search to find the k best translations, we do not use a usual log-probability but one normalized with respect to the length of the translation. This prevents the RNN decoder from favoring shorter translations, behavior which was observed earlier in, e.g., (Graves, 2013).

[Cho et al., 2014]

$$s(Y,X) = \log(P(Y|X))/lp(Y) + cp(X;Y)$$

$$lp(Y) = \frac{(5+|Y|)^\alpha}{(5+1)^\alpha}$$

$$cp(X;Y) = \beta * \sum_{i=1}^{|X|} \log(\min(\sum_{j=1}^{|Y|} p_{i,j}, 1.0)),$$

[Wu et al., 2016]

Oversmoothing in the wild

Post-hoc fixes

- Learned length penalty [Murray & Chiang, 2018]

Finally, some systems add a constant *word reward* (He et al., 2016):

$$s'(e) = s(e) + \gamma m.$$

If $\gamma = 0$, this reduces to the baseline model. The advantage of this simple reward is that it can be computed on partial translations, making it easier to integrate into beam search.

If we approximate the expectation using the mode of the distribution, we get

$$\frac{\partial L}{\partial \gamma} \approx -|e^*| + |\hat{e}|$$

where $\hat{e}$ is the 1-best translation. Then the stochastic gradient descent update is just the familiar perceptron rule:

$$\gamma \leftarrow \gamma + \eta\,(|e^*| - |\hat{e}|),$$

Oversmoothing in the wild

Post-hoc fixes

- Length penalty is a bandage and does not get to the heart of the problem.

> However, none of these papers directly solves the mystery outlined in Section 1, i.e., why doesn't maximimum likelihood training already shift probability away from emtpy output candidates, toward appropriate-length ones?

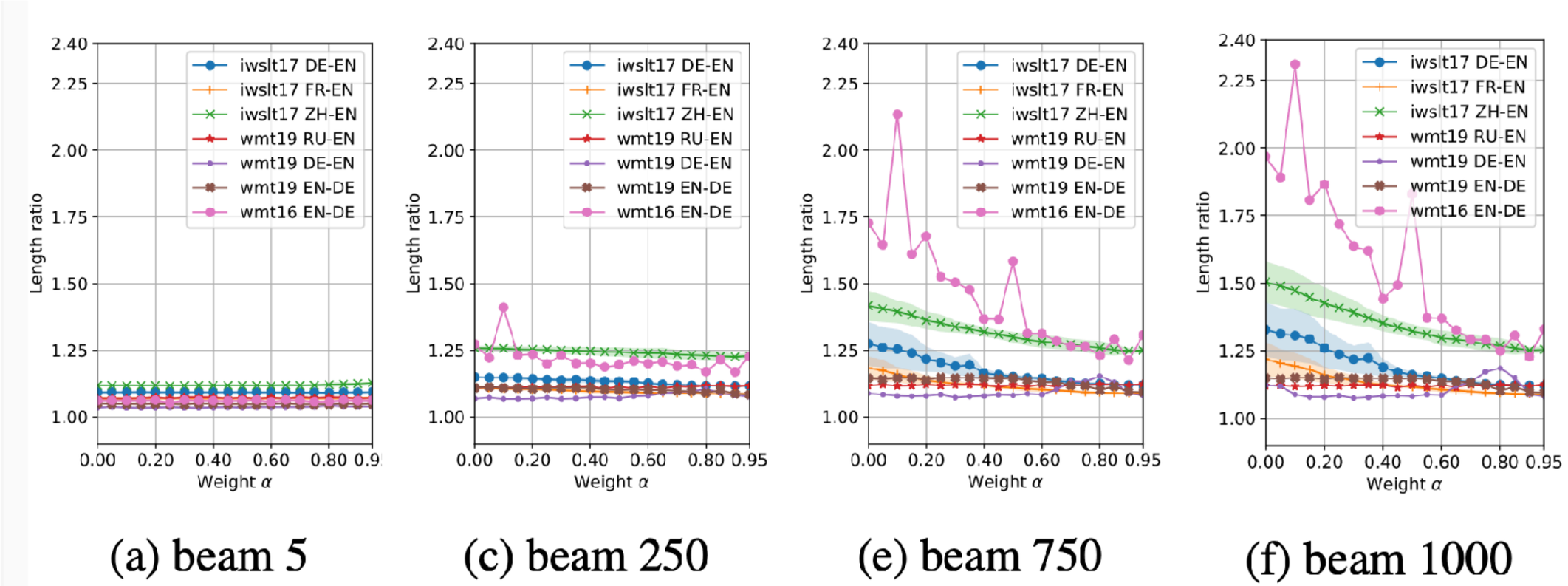
Shi et al. [2020]

- Of course.. we just did 😉

Let's now come back to our method.

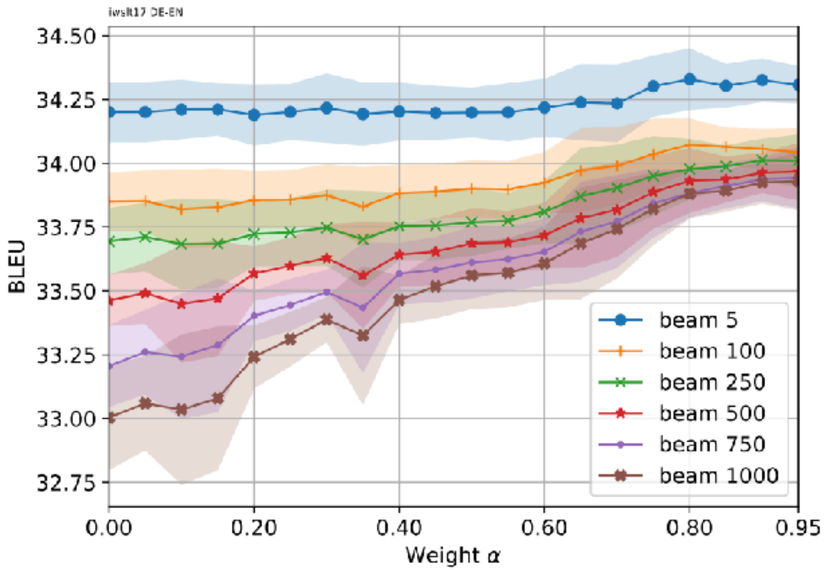
Length degeneracy disappears

- Strong regularization of oversmoothing removes the issue of length degeneracy
- Even *without* using any length penalty during beam search

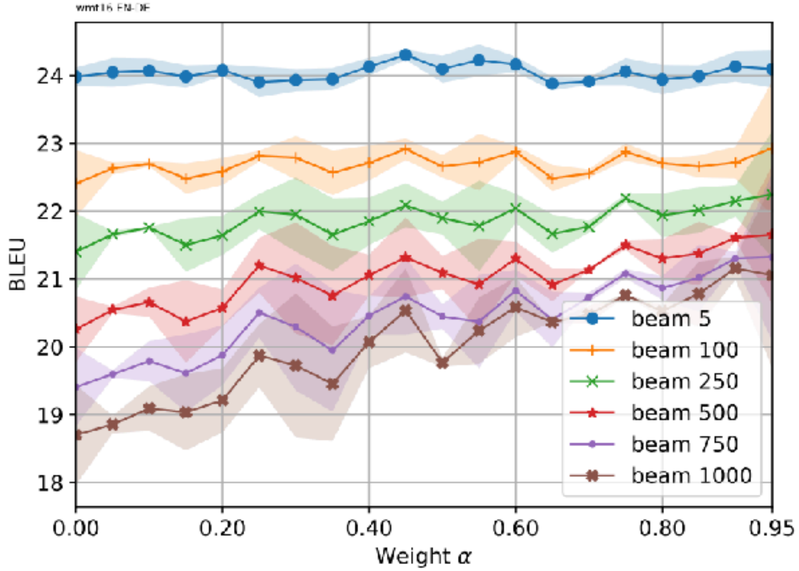


Oversmoothing vs. translation quality

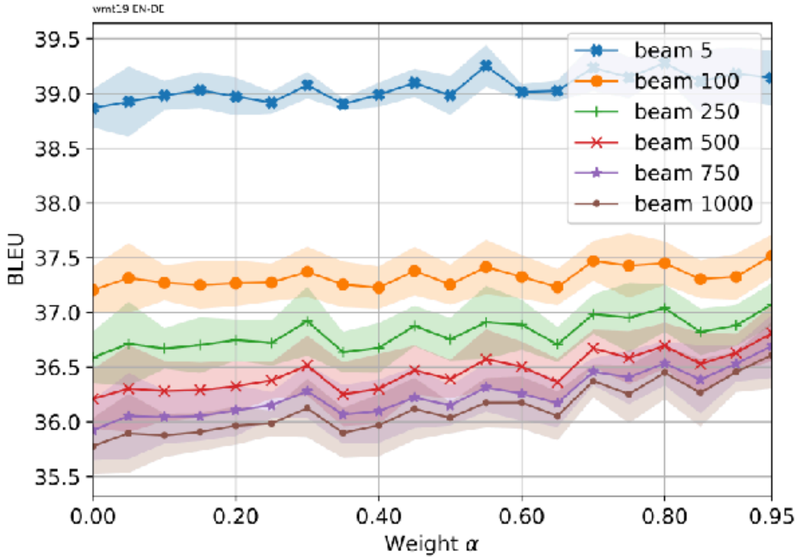
- With highly approximate search (beam5), no impact on translation quality.
- With less approximate search (beam1000), we observe significant improvement in translation quality with lessened oversmoothing
  - Mostly from ruling out unreasonably short translations.



(a) IWSLT'17 DE→EN



(b) WMT'16 EN→DE



(c) WMT'19 EN→DE

Problem solved … ?

Obviously not … :(

Oversmoothing

Positive news

- Carefully determined/defined the issue of oversmoothing:

$$r(y) = \frac{1}{T}\sum_{t=1}^{T} I\left(p(\langle \text{eos} \rangle \,|\, y_{<t}) > \prod_{t'=t}^{T} p(y_{t'} \,|\, y_{<t'}) \right)$$

- Carefully designed a loss function to alleviate this issue:

$$o(y) = \frac{1}{T}\sum_{t=1}^{T} \max\left(0, m + \log p(\langle \text{eos} \rangle \,|\, y_{<t}) - \log \prod_{t'=t}^{T} p(y_{t'} \,|\, y_{<t'}) \right)$$

- Experimentally demonstrated the effectiveness of this oversmoothing loss.

Oversmoothing

Negative news

- BUT, translation quality has not improved.
- AND, better translations are still found by more approximate search
  - Beam search with a small beam

- **Mystery continues** …
  - How is **beam search** finding those **high-quality** but **not necessarily most probable** translations from our model?

Fin.