Neural Sequence Generation

He He

NEW YORK UNIVERSITY

September 27, 2023

Sequence generation

- Text classification: $h : \mathcal{V}^n \rightarrow \{0, \ldots, K\}$

Sequence generation

- Text classification: $h : \mathcal{V}^n \rightarrow \{0, \ldots, K\}$
- Sequence generation: $h : \mathcal{V}_{\text{in}}^n \rightarrow \mathcal{V}_{\text{out}}^m$
  - Summarization: document to summary
  - Open-domain dialogue: context to response
  - Parsing: sentence to linearized trees
  - In general: text to text

Sequence generation

- Text classification: $h : \mathcal{V}^n \rightarrow \{0, \ldots, K\}$
- Sequence generation: $h : \mathcal{V}_{\text{in}}^n \rightarrow \mathcal{V}_{\text{out}}^m$
    - Summarization: document to summary
    - Open-domain dialogue: context to response
    - Parsing: sentence to linearized trees
    - In general: text to text

Sequence generation

- Text classification: $h : \mathcal{V}^n \to \{0, \dots, K\}$
- Sequence generation: $h : \mathcal{V}_{\text{in}}^n \to \mathcal{V}_{\text{out}}^m$
  - Summarization: document to summary
  - Open-domain dialogue: context to response
  - Parsing: sentence to linearized trees
  - In general: text to text

Main difference (and challenge) is that the output space is much larger.

Reduce generation to classification

Setup:

- Input: $x \in \mathcal{V}_{\text{in}}^{n}$, e.g. *Le Programme a ate mis en application*
- Output: $y \in \mathcal{V}_{\text{out}}^{m}$, e.g., *The program has been implemented*

Reduce generation to classification

Setup:

- Input: $x \in \mathcal{V}_{\text{in}}^{n}$, e.g. *Le Programme a ate mis en application*
- Output: $y \in \mathcal{V}_{\text{out}}^{m}$, e.g., *The program has been implemented*

Consider a probabilistic model $p(y \mid x)$

- Can we reduce it to classification (think logistic regression)?

Reduce generation to classification

Setup:

- Input: $x \in \mathcal{V}_{\text{in}}^n$, e.g. *Le Programme a ate mis en application*
- Output: $y \in \mathcal{V}_{\text{out}}^m$, e.g., *The program has been implemented*

Consider a probabilistic model $p(y \mid x)$

- Can we reduce it to classification (think logistic regression)?
- Decompose the problem using **chain rule of probability**

$$\begin{aligned} p(y \mid x) &= p(y_1 \mid x)p(y_2 \mid y_1, x) \ldots p(y_m \mid y_{m-1}, \ldots, y_1, x) \\ &= \prod_{i=1}^{m} p(y_i \mid y_{<i}, x) \end{aligned}$$

Reduce generation to classification

Setup:

- Input: $x \in \mathcal{V}_{\text{in}}^{n}$, e.g. *Le Programme a ate mis en application*
- Output: $y \in \mathcal{V}_{\text{out}}^{m}$, e.g., *The program has been implemented*

Consider a probabilistic model $p(y \mid x)$

- Can we reduce it to classification (think logistic regression)?
- Decompose the problem using **chain rule of probability**

$$\begin{aligned} p(y \mid x) &= p(y_1 \mid x)p(y_2 \mid y_1, x) \ldots p(y_m \mid y_{m-1}, \ldots, y_1, x) \\ &= \prod_{i=1}^{m} p(y_i \mid y_{<i}, x) \end{aligned}$$

- We only need to model the next word distribution $p(y_i \mid y_{<i}, x)$ now.

Reduce generation to classification

We want to model the next word distribution $p(y_i \mid y_{<i}, x)$.

- Input: a sequence of tokens (prefix and input)

Reduce generation to classification

We want to model the next word distribution $p(y_i \mid y_{<i}, x)$.

- Input: a sequence of tokens (prefix and input)
- Output: the next word from the output vocabulary

Reduce generation to classification

We want to model the next word distribution $p(y_i \mid y_{<i}, x)$.

- Input: a sequence of tokens (prefix and input)
- Output: the next word from the output vocabulary
- We have reduced it to a classification problem.

Reduce generation to classification

We want to model the next word distribution $p(y_i \mid y_{<i}, x)$.

- Input: a sequence of tokens (prefix and input)
- Output: the next word from the output vocabulary
- We have reduced it to a classification problem.

Reduce generation to classification

We want to model the next word distribution $p(y_i \mid y_{<i}, x)$.

- Input: a sequence of tokens (prefix and input)
- Output: the next word from the output vocabulary
- We have reduced it to a classification problem.

We can use an RNN to model $p(y_i \mid y_{<i}, x)$.

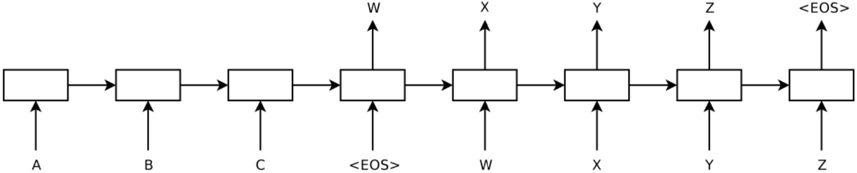


Figure: From Sequence to Sequence Learning with Neural Networks [Sutskever et al., 2014]

The encoder-decoder architecture

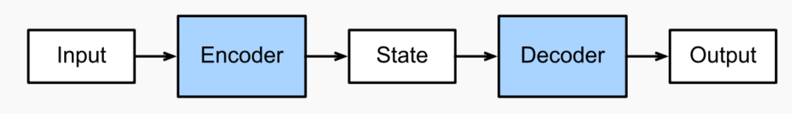


Figure: 10.6.1 from d2l.ai

Model the input (e.g., French) and the output (e.g., English) separately.

The encoder-decoder architecture

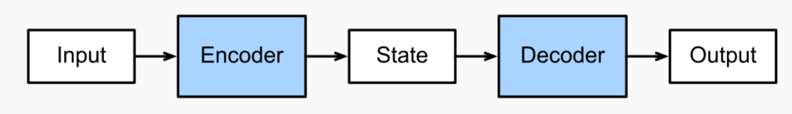


Figure: 10.6.1 from d2l.ai

Model the input (e.g., French) and the output (e.g., English) separately.

- The **encoder** reads the input:

$$\mathrm{Encoder}(x_1, \ldots, x_n) = [h_1, \ldots, h_n]$$

where $h_i \in \mathbb{R}^d$

The encoder-decoder architecture

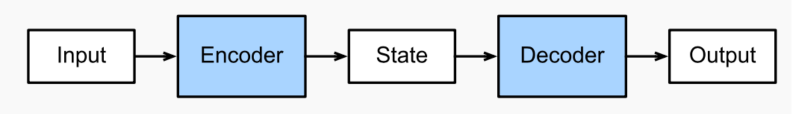


Figure: 10.6.1 from d2l.ai

Model the input (e.g., French) and the output (e.g., English) separately.

- The **encoder** reads the input:

$$\mathrm{Encoder}(x_1, \ldots, x_n) = [h_1, \ldots, h_n]$$

  where $h_i \in \mathbb{R}^d$
- The **decoder** writes the output:

$$\mathrm{Decoder}(h_1, \ldots, h_n) = [y_1, \ldots, y_m]$$

.

RNN encoder-decoder model

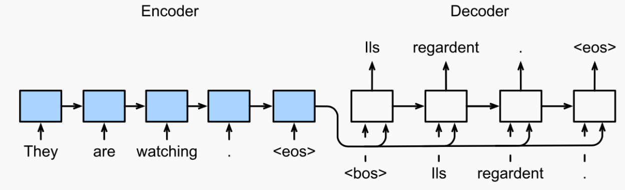


Figure: 10.7.1 from d2l.ai

- The **encoder** embeds the input recurrently and produce a **context vector**

$$h_t = \mathrm{RNNEncoder}(x_t, h_{t-1}), \quad c = f(h_1, \dots, h_n)$$

RNN encoder-decoder model

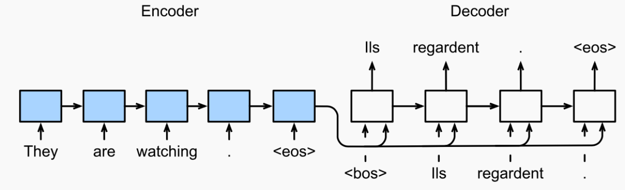


Figure: 10.7.1 from d2l.ai

- The **encoder** embeds the input recurrently and produce a **context vector**

$$h_t = \mathrm{RNNEncoder}(x_t, h_{t-1}), \quad c = f(h_1, \dots, h_n)$$

- The **decoder** produce the output state recurrently and map it to a distribution over tokens

$$s_t = \mathrm{RNNDecoder}([y_{t-1}; c], s_{t-1}), \quad p(y_t \mid y_{<t}, c) = \mathrm{softmax}(\mathrm{Linear}(s_t))$$

Bi-directional RNN encoder

The [Forbes]$_{??}$ building is at 60 Fifth Ave.

Bi-directional RNN encoder

The [Forbes]$_{??}$ building is at 60 Fifth Ave.

Each hidden state should summarize both left and right context

Bi-directional RNN encoder

The [Forbes]$_{??}$ building is at 60 Fifth Ave.

Each hidden state should summarize both left and right context

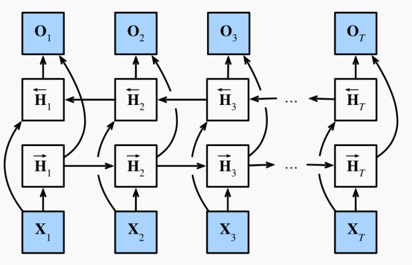


Figure: 10.4.1 from d2l.ai

- Use two RNNs, one encode from left to right, the other from right to left
- Concatenate hidden states from the two RNNs

$$h_t = [\overleftarrow{h_t}; \overrightarrow{h_t}]$$
$$o_t = Wh_t + b$$

Multilayer RNN

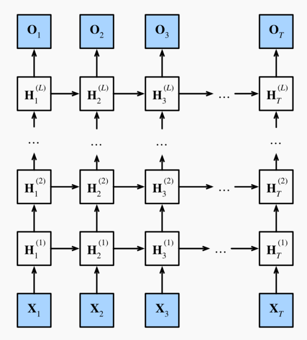
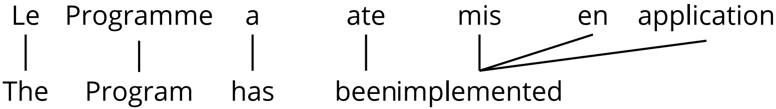


Figure: 10.3.1 from d2l.ai

- Improve model capacity (scaling up)
- Inputs to layer 1 are words
- Inputs to layer l are outputs from layer $l - 1$
- Typically 2–4 layers

Encoder-decoder attention

Motivation: should we use the same context vector for each decoding step?



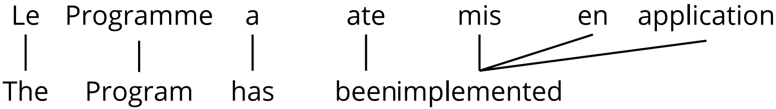
Le Programme a ate mis en application

The Program has beenimplemented

We may want to “look at” different parts of the input during decoding.

Encoder-decoder attention

Motivation: should we use the same context vector for each decoding step?



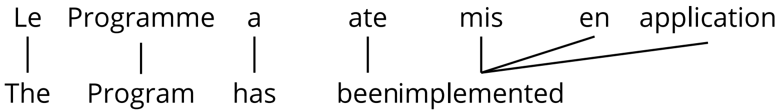
We may want to “look at” different parts of the input during decoding.

Think the database analogy:

- Query: decoder states s_{t-1}

Encoder-decoder attention

Motivation: should we use the same context vector for each decoding step?



Le Programme a ate mis en application

The Program has beenimplemented

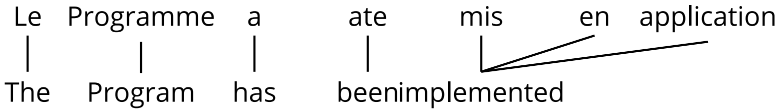
We may want to “look at” different parts of the input during decoding.

Think the database analogy:

- Query: decoder states s_{t-1}
- Key: encoder states $h_1, \ldots, h_n$

Encoder-decoder attention

Motivation: should we use the same context vector for each decoding step?


Le Programme a ate mis en application
The Program has beenimplemented

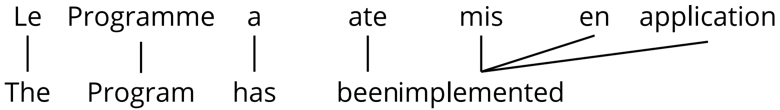
We may want to “look at” different parts of the input during decoding.

Think the database analogy:

- Query: decoder states s_{t-1}
- Key: encoder states $h_1, \ldots, h_n$
- Value: encoder states $h_1, \ldots, h_n$

Encoder-decoder attention

Motivation: should we use the same context vector for each decoding step?



Le Programme a ate mis en application

The Program has beenimplemented

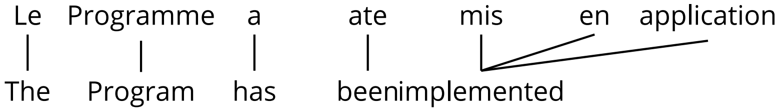
We may want to “look at” different parts of the input during decoding.

Think the database analogy:

- Query: decoder states s_{t-1}
- Key: encoder states $h_1, \dots, h_n$
- Value: encoder states $h_1, \dots, h_n$
- Attention context: $c_t = \sum_{i=1}^{n} \alpha(s_{t-1}, h_i) h_i$

Encoder-decoder attention

Motivation: should we use the same context vector for each decoding step?



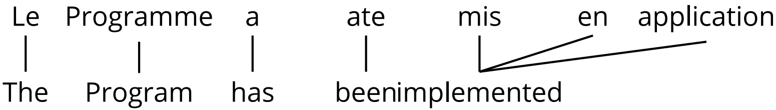
We may want to “look at” different parts of the input during decoding.

Think the database analogy:

- Query: decoder states s_{t-1}
- Key: encoder states $h_1, \ldots, h_n$
- Value: encoder states $h_1, \ldots, h_n$
- Attention context: $c_t = \sum_{i=1}^{n} \alpha(s_{t-1}, h_i) h_i$
- Next state: $s_t = \mathrm{RNNDecoder}(\left[y_{t-1}; c_t\right], s_{t-1})$

Encoder-decoder attention

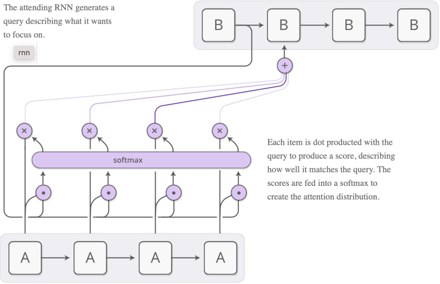
Motivation: should we use the same context vector for each decoding step?



We may want to “look at” different parts of the input during decoding.

Think the database analogy:

- Query: decoder states s_{t-1}
- Key: encoder states $h_1, \ldots, h_n$
- Value: encoder states $h_1, \ldots, h_n$
- Attention context: $c_t = \sum_{i=1}^{n} \alpha(s_{t-1}, h_i) h_i$
- Next state: $s_t = \mathrm{RNNDecoder}([y_{t-1}; c_t], s_{t-1})$



The attending RNN generates a query describing what it wants to focus on.

Each item is dot producted with the query to produce a score, describing how well it matches the query. The scores are fed into a softmax to create the attention distribution.

Summary so far

The outputs of an encoder can be used by linear classifiers for classification, sequence labeling etc.

A decoder is used to generate a sequence of symbols.

RNN encoder decoder model:

- Basic unit is an RNN (or its variants like LSTM)
- Make it more expressive: bi-directional, multilayer RNN
- Encoder-decoder attention helps the model learn input-output dependencies more easily
- Bi-directional LSTM is the go-to architecture for NLP tasks until around 2017

Transformer encoder decoder model

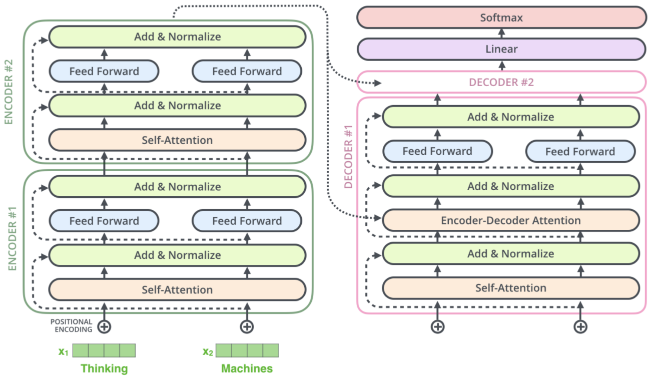


Figure: From illustrated transformer

- Stack the tranformer block (typically 12–24 layers)
- Decoder has an additional encoder-decoder multi-head attention layer

Transformer encoder decoder model

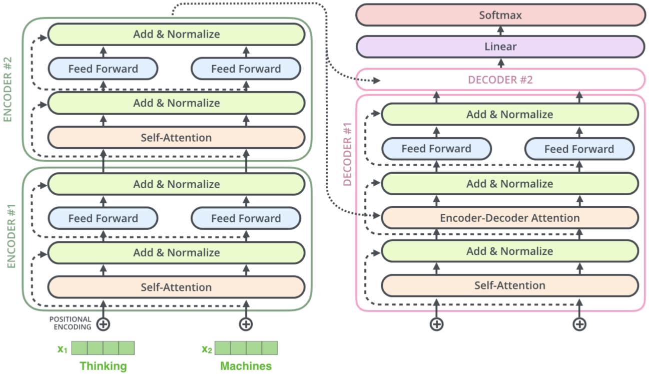


Figure: From illustrated transformer

- Stack the tranformer block (typically 12–24 layers)
- Decoder has an additional encoder-decoder multi-head attention layer

Impact on NLP

- Initially designed for sequential data and obtained SOTA results on MT
- Replaced recurrent models (e.g. LSTM) on many tasks
- Enabled large-scale training which led to pre-trained models such as BERT and GPT-2

Next, training and inference of encoder-decoder models.

Training

Maximum likelihood estimation:

$$\max \sum_{(x,y)\in\mathcal{D}} \sum_{j=1}^{m} \log p(y_j \mid y_{<j}, x; \theta)$$

Training

Maximum likelihood estimation:

$$\max \sum_{(x,y)\in\mathcal{D}} \sum_{j=1}^{m} \log p(y_j \mid y_{<j}, x; \theta)$$

What is the prefix $y_{<j}$?

Training

Maximum likelihood estimation:

$$\max \sum_{(x,y)\in\mathcal{D}} \sum_{j=1}^{m} \log p(y_j \mid y_{<j}, x; \theta)$$

What is the prefix $y_{<j}$?

Use the groundtruth prefix (**teacher forcing**)

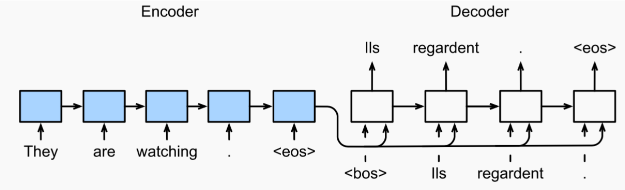


Figure: 10.7.3 from d2l.ai

Decoder attention masking

Recall that the output of self-attention depends on all tokens $y_1, \ldots y_m$.

But the decoder is supposed to model $p(y_t \mid y_{<t}, x)$.

It should not look at the “future” ($y_{t+1}, \ldots, y_m$)!

Decoder attention masking

Recall that the output of self-attention depends on all tokens $y_1, \ldots y_m$.

But the decoder is supposed to model $p(y_t \mid y_{<t}, x)$.

It should not look at the “future” ($y_{t+1}, \ldots, y_m$)!

How do we fix the decoder self-attention?

- Mathematically, changing the input values and keys suffices.
- Practically, set $a(s_i, s_j)$ to $-$inf for all $j > i$ and for $i = 1, \ldots, m$.
    - The attention matrix is a lower-triangular matrix.

Inference

Suppose we have a trained model $p(y \mid x; \theta)$.

The model defines a probability distribution over all possible sequences.

But we want to output a single sequence.

The **decoding** problem: How do we predict a sequence from the model?

Inference

Argmax decoding:

$$\hat{y} = \arg\max_{y \in \mathcal{V}_{\text{out}}^{n}} p(y \mid x; \theta)$$

- Return the most likely sequence
- But exact search is intractable

Inference

Argmax decoding:

$$\hat{y} = \underset{y \in \mathcal{V}_{\text{out}}^{n}}{\arg\max}\, p(y \mid x; \theta)$$

- Return the most likely sequence
- But exact search is intractable

Approximate search:

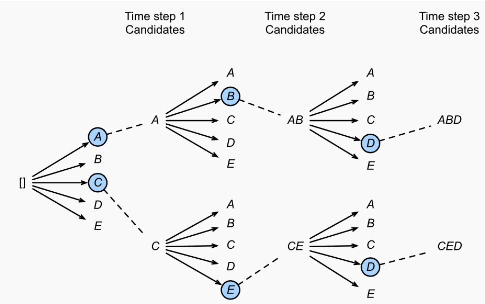
- **Greedy decoding**: return the most likely symbol at each step

$$y_t = \underset{y \in \mathcal{V}_{\text{out}}}{\arg\max}\, p(y \mid x, \hat{y}_{<t}; \theta)$$

Approximate decoding: beam search

Beam search: maintain k (beam size) highest-scored partial solutions at every step

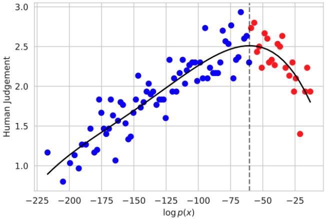
Example: $|\mathcal{V}| = 5, k = 2$

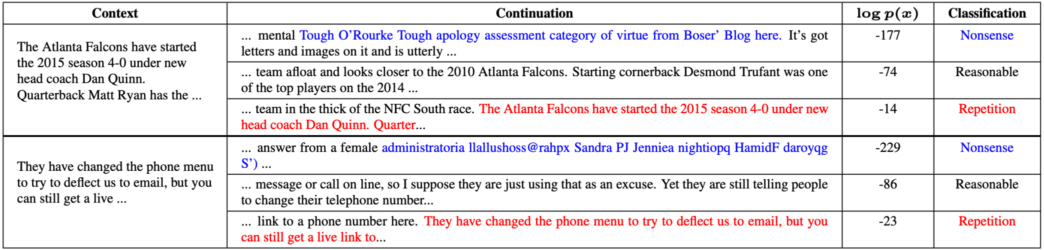


- At each step, rank symbols by log probability of the partial sequence
- Keep the top-k symbol out of all possible continuations
- Save **backpointer** to the previous state

Is argmax the right decoding objective?

High likelihood can be correlated with low quality outputs! [Zhang et al., 2020]





Context	Continuation	$\log p(x)$	Classification
The Atlanta Falcons have started the 2015 season 4-0 under new head coach Dan Quinn. Quarterback Matt Ryan has the ...	... mental Tough O'Rourke Tough apology assessment category of virtue from Boser' Blog here. It's got letters and images on it and is utterly ...	-177	Nonsense
	... team afloat and looks closer to the 2010 Atlanta Falcons. Starting cornerback Desmond Trufant was one of the top players on the 2014 ...	-74	Reasonable
	... team in the thick of the NFC South race. The Atlanta Falcons have started the 2015 season 4-0 under new head coach Dan Quinn. Quarter...	-14	Repetition
They have changed the phone menu to try to deflect us to email, but you can still get a live ...	... answer from a female administratoria llallushoss@rahpx Sandra PJ Jenniea nightiopq HamidF daroyqg S') ...	-229	Nonsense
	... message or call on line, so I suppose they are just using that as an excuse. Yet they are still telling people to change their telephone number...	-86	Reasonable
	... link to a phone number here. They have changed the phone menu to try to deflect us to email, but you can still get a live link to...	-23	Repetition

Is argmax the right decoding objective?

In practice, argmax decoding has been observed to lead to

- Repetitive generations, e.g., "..., was conducted by researchers from the Universidad Nacional Autonoma de Mexico (UNAM) and the Universidad Nacional Autonoma de Mexico (UNAM/Universidad Nacional Autonoma de Mexico/Universidad Nacional Autonoma de Mexico/Universidad Nacional Autonoma..."
- Empty or extremely short translations with large beam size in MT

Is argmax the right decoding objective?

In practice, argmax decoding has been observed to lead to

- Repetitive generations, e.g., "..., was conducted by researchers from the Universidad Nacional Autonoma de Mexico (UNAM) and the Universidad Nacional Autonoma de Mexico (UNAM/Universidad Nacional Autonoma de Mexico/Universidad Nacional Autonoma de Mexico/Universidad Nacional Autonoma..."
- Empty or extremely short translations with large beam size in MT

Hypotheses:

- Models don't fit the data well
  But problem doesn't go away with larger model and data
- Distribution shift during inference (more on this later)
  Need formulation and evidence

Sampling-based decoding

If we have learned a perfect $p(y \mid x)$, shouldn't we just sample from it?

Sampling-based decoding

If we have learned a perfect $p(y \mid x)$, shouldn't we just sample from it?

Sampling is easy for autoregressive models:

- While output is not `EOS`
  - Sample next word from $p(\cdot \mid \text{prefix}, \text{input}; \theta)$
  - Append the word to prefix

Sampling-based decoding

If we have learned a perfect $p(y \mid x)$, shouldn't we just sample from it?

Sampling is easy for autoregressive models:

- While output is not `EOS`
  - Sample next word from $p(\cdot \mid \text{prefix}, \text{input}; \theta)$
  - Append the word to prefix

Standard sampling often produces non-sensical sentences:

> They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town, and they speak huge, beautiful, paradisiacal Bolivian linguistic thing.

Idea: modify the learned distrubtion p_θ before sampling to avoid bad generations

Tempered sampling

Intuition: concentrate probability mass on highly likely sequences

Scale scores (from the linear layer) before the softmax layer:

$$\begin{aligned} p(y_t = w \mid y_{<t}, x) &\propto \exp\left(\mathrm{score}(w)\right) \\ q(y_t = w \mid y_{<t}, x) &\propto \exp\left(\mathrm{score}(w)/T\right) \quad \text{where } T \in (0, +\infty) \end{aligned}$$

Tempered sampling

Intuition: concentrate probability mass on highly likely sequences

Scale scores (from the linear layer) before the softmax layer:

$$p(y_t = w \mid y_{<t}, x) \propto \exp\left(\mathrm{score}(w)\right)$$
$$q(y_t = w \mid y_{<t}, x) \propto \exp\left(\mathrm{score}(w)/T\right) \quad \text{where } T \in (0, +\infty)$$

- What happends when $T \to 0$ and $T \to +\infty$?
- Does it change the rank of y according to likelihood?
- Typically we chooose $T \in (0, 1)$, which makes the distribution more peaky.

Truncated sampling

Another way to focus on highly likely sequences: truncate the tail of the distribution

Top-k sampling:

- Rank all tokens $w \in \mathcal{V}$ by $p(y_t = w \mid y_{<t}, x)$
- Only keep the top k of those and renormalize the distribution

Effect of k:

- Large k: more diverse but possibly degenerate outputs
- Small k: more generic but safe outputs

Truncated sampling

Which *k* to choose?

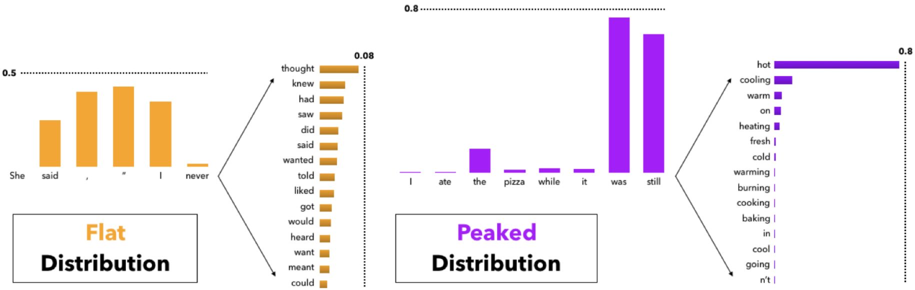


Figure: From the nucleus sampling paper by Holtzman et al., 2020

Using a single *k* on different next word distributions may be suboptimal

Truncated sampling

Top-p sampling:

- Rank all tokens $w \in \mathcal{V}$ by $p(y_t = w \mid y_{<t}, x)$
- Keep only tokens in the top p probability mass and renormalize the distribution
- The corresponding k is dynamic:
  - Start with $k = 1$, increment until the cumulative probability mass $> p$

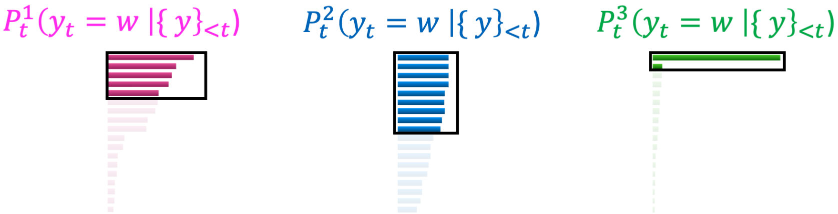


Figure: From Xiang Li's slides

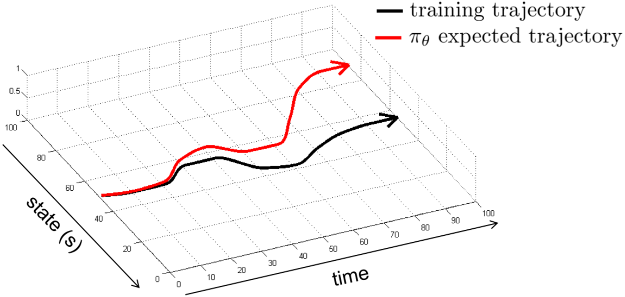
Decoding in practice

- Can combine different tricks (e.g., temperature + beam search, temperature + top-k)
- Use beam search with small beam size for tasks where there exists a correct answer, e.g. machine translation, summarization
- Use top-k or top-p for open-ended generation, e.g. story generation, chit-chat dialogue, continuation from a prompt
- As models getting better/larger, sampling-based methods tend to work better

Exposure bias

Problem with teacher forcing:

- During training, the model only sees groundtruth prefix
- During inference, the model sees generated prefix, which may deviate from the training prefix distribution
- When this happends, the model behavior is underspecified.



Exposure bias

Solutions:

- Avoid deviating from the training prefix distribution
  - Better modeling: reduce errors at each step
  - Better decoding: stay within the high likelihood region (later)

Exposure bias

Solutions:

- Avoid deviating from the training prefix distribution
  - Better modeling: reduce errors at each step
  - Better decoding: stay within the high likelihood region (later)
- Teaching the model how to behave on out-of-distribution prefix
  - Better learning: updating models based on the goodness of the *generated* sequence

Exposure bias

Solutions:

- Avoid deviating from the training prefix distribution
  - Better modeling: reduce errors at each step
  - Better decoding: stay within the high likelihood region (later)
- Teaching the model how to behave on out-of-distribution prefix
  - Better learning: updating models based on the goodness of the *generated* sequence

    additional supervision required

    computationally more expensive