Neural Sequence Generation

He He

NEW YORK UNIVERSITY

Febuary 12, 2024

Table of Contents

Last week

- We have seen two families of models for sequences modeling: **RNNs** and **Transformers**
- They are often called **encoders**: take a sequence of tokens and output a sequence of embeddings
- Each embedding is a **contextualized representation** of the token
- We can then use the embeddings for classification or sequence labeling
- Three building blocks for encoders:
  - Multilayer perceptron
  - Recurrent neural networks
  - Self-attention

  Which one is simplest in terms of computation?

Table of Contents

Transformer block

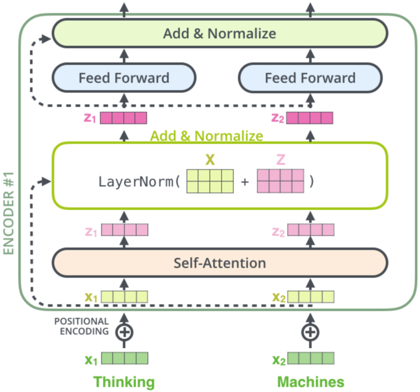


Figure: From The Illustrated Transformer

- Multi-head self-attention
  - Compute contextualized representations
- Positional encoding
  - Represent the order of tokens
- Residual connection and layer normalization
  - More efficient and stable optimization

Recap: multi-head self-attention

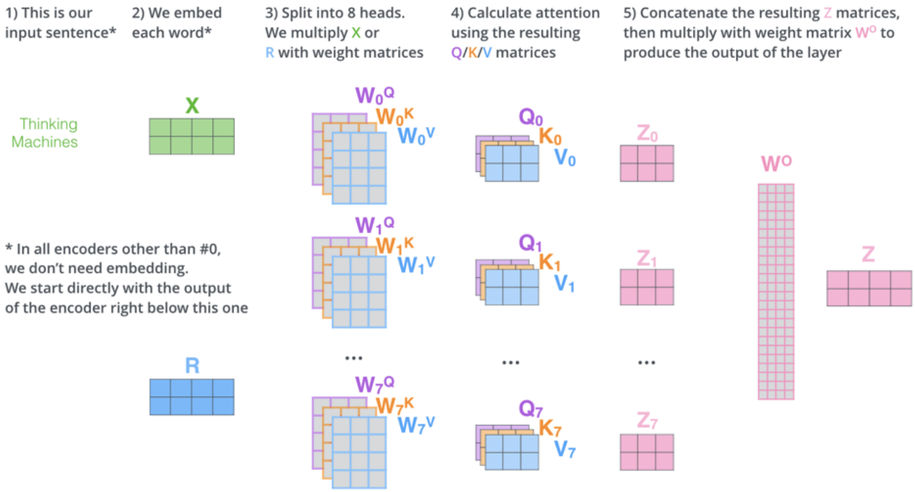


Figure: From The Illustrated Transformer

Recap: sinusoidal position embedding

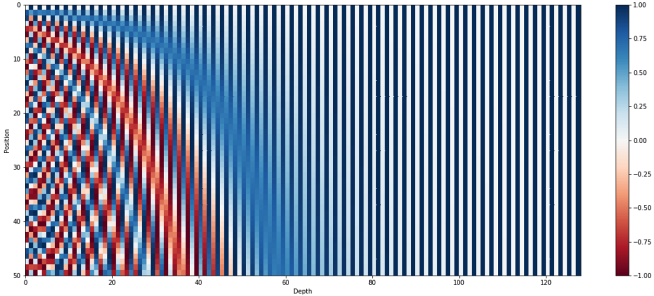
Intuition: continuous approximation of binary encoding of positions (integers)

0 :	0	0	0	0
1 :	0	0	0	1
2 :	0	0	1	0
3 :	0	0	1	1
4 :	0	1	0	0
5 :	0	1	0	1
6 :	0	1	1	0
7 :	0	1	1	1

Recap: sinusoidal position embedding

Intuition: continuous approximation of binary encoding of positions (integers)

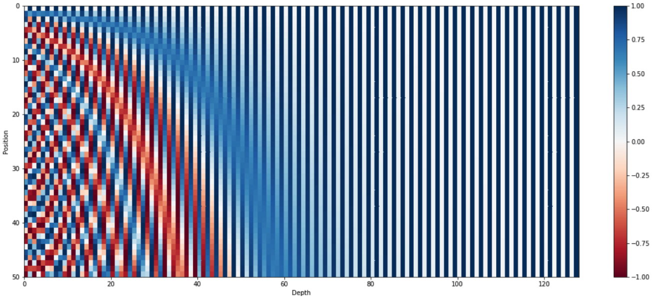
0 :	0	0	0	0
1 :	0	0	0	1
2 :	0	0	1	0
3 :	0	0	1	1
4 :	0	1	0	0
5 :	0	1	0	1
6 :	0	1	1	0
7 :	0	1	1	1



Recap: sinusoidal position embedding

Intuition: continuous approximation of binary encoding of positions (integers)

0 :	0	0	0	0
1 :	0	0	0	1
2 :	0	0	1	0
3 :	0	0	1	1
4 :	0	1	0	0
5 :	0	1	0	1
6 :	0	1	1	0
7 :	0	1	1	1



$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

Figure: From Amirhossein Kazemnejad's Blog

- Each row is an embedding for a particular position
- Each column is a sinusoidal wave with a particular frequency

Residual connection

Motivation:

- Gradient explosion/vanishing is not RNN-specific!
- It happens to all deep networks (which are hard to optimize).

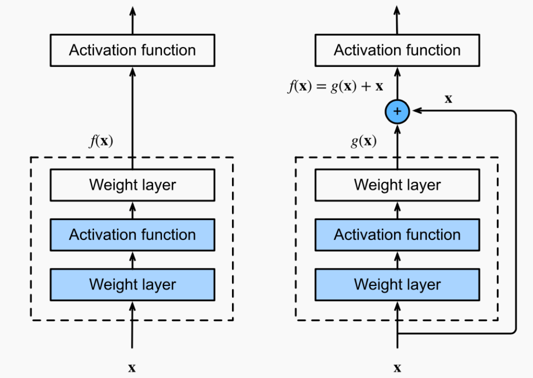
Residual connection

Motivation:

- Gradient explosion/vanishing is not RNN-specific!
- It happens to all deep networks (which are hard to optimize).
- In principle, a deep network can always represent a shallow network (by setting higher layers to identity functions), thus it should be at least as good as the shallow network.
- For some reason, deep neural networks are bad at learning identity functions.
- How can we make it easier to recover the shallow solution?

Residual connection

Solution: Deep Residual Learning for Image Recognition [He et al., 2015]



Without residual connection: learn the identity function $f(x) = x$.

With residual connection: learn $g(x) = 0$ (easier).

Layer normalization

- Normalize each input sample to zero mean and unit variance [Ba et al., 2016]
- Let $x = (x_1, \ldots, x_d)$ be the input vector (e.g., word embedding, previous layer output)

$$\mathrm{LayerNorm}(x) = \frac{x - \hat{\mu}}{\hat{\sigma}},$$

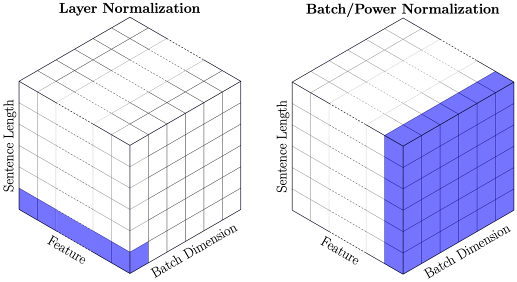
$$\text{where } \hat{\mu} = \frac{1}{d}\sum_{i=1}^{d} x_i, \quad \hat{\sigma}^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \hat{\mu})^2$$

Layer normalization

- Normalize each input sample to zero mean and unit variance [Ba et al., 2016]
- Let $x = (x_1, \ldots, x_d)$ be the input vector (e.g., word embedding, previous layer output)

$$\mathrm{LayerNorm}(x) = \frac{x - \hat{\mu}}{\hat{\sigma}},$$

$$\text{where } \hat{\mu} = \frac{1}{d}\sum_{i=1}^{d} x_i, \quad \hat{\sigma}^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \hat{\mu})^2$$

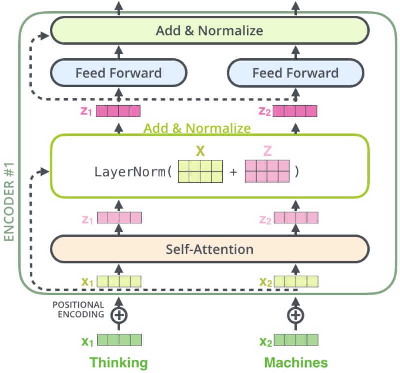


- Independent of train/inference and batch size
- Robust to varying sequence length in a batch

Why do we need layer normalization

- Main reason: **training stability** for *deep* neural networks (avoiding NaN, diverging loss, etc.)
- Sources of instability:
  - Matrix multiplication:
    $$h^{(l)} = W^{(l-1)} h^{(l-1)}$$
    Small changes accumulates multiplicatively through the layers.
  - Residual connection:
    $$h^{(l)} = h^{(l-1)} + f(h^{(l-1)})$$
    Small changes accumulates multiplicatively through the layers.
  - Softmax saturation:
    $$\mathrm{softmax}(x)_i = \frac{e^{x_i}}{\sum_i e^{x_i}}$$
    Large x_i drives vanishing gradients

Putting everything together



- Add (residual connection) & Normalize (layer normalization) for the output of self-attention and FFN (post-LN)

Pre-layer normalization

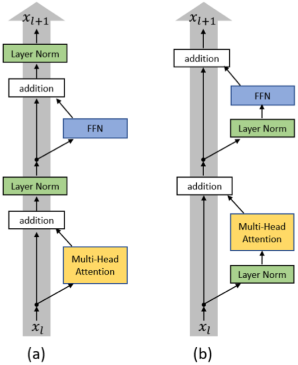
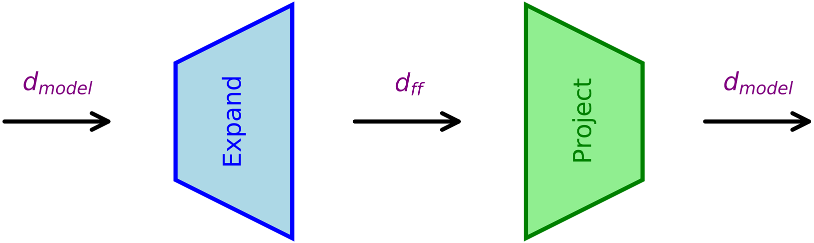


Figure: From [Xiong et al. 2020]

- Post-LN: normalize the output of each layer
- Pre-LN: normalize the input of each layer
- Use either or both

Putting everything together



- Same FFN applied to each embedding
- Two layers: first layer expands the dimension (e.g., $d \to 4d$), second layer projects it back (e.g., $4d \to d$)

Table of Contents

Sequence generation

- Given a sequence of contextualized embeddings, we can do classification.

Sequence generation

- Given a sequence of contextualized embeddings, we can do classification.
- What if we want to predict a sequence of tokens from the input sequence (e.g., machine translation)?

Sequence generation

- Given a sequence of contextualized embeddings, we can do classification.
- What if we want to predict a sequence of tokens from the input sequence (e.g., machine translation)?
- Sequence classification: $h : \mathcal{V}^n \to \{0, \ldots, K\}$
  - Sentiment classification

Sequence generation

- Given a sequence of contextualized embeddings, we can do classification.
- What if we want to predict a sequence of tokens from the input sequence (e.g., machine translation)?
- Sequence classification: $h : \mathcal{V}^n \to \{0, \ldots, K\}$
  - Sentiment classification
- Sequence labeling: $h : \mathcal{V}^n \to \{0, \ldots, K\}^n$
  - Part-of-speech tagging

Sequence generation

- Given a sequence of contextualized embeddings, we can do classification.
- What if we want to predict a sequence of tokens from the input sequence (e.g., machine translation)?
- Sequence classification: $h : \mathcal{V}^n \rightarrow \{0, \ldots, K\}$
  - Sentiment classification
- Sequence labeling: $h : \mathcal{V}^n \rightarrow \{0, \ldots, K\}^n$
  - Part-of-speech tagging
- Sequence generation: $h : \mathcal{V}_{\text{in}}^n \rightarrow \mathcal{V}_{\text{out}}^m$
  - Summarization: document to summary
  - In general: sequence to sequence

Sequence generation

- Given a sequence of contextualized embeddings, we can do classification.
- What if we want to predict a sequence of tokens from the input sequence (e.g., machine translation)?
- Sequence classification: $h : \mathcal{V}^n \to \{0, \ldots, K\}$
  - Sentiment classification
- Sequence labeling: $h : \mathcal{V}^n \to \{0, \ldots, K\}^n$
  - Part-of-speech tagging
- Sequence generation: $h : \mathcal{V}_{\text{in}}^n \to \mathcal{V}_{\text{out}}^m$
  - Summarization: document to summary
  - In general: sequence to sequence

Sequence generation

- Given a sequence of contextualized embeddings, we can do classification.
- What if we want to predict a sequence of tokens from the input sequence (e.g., machine translation)?
- Sequence classification: $h : \mathcal{V}^n \to \{0, \ldots, K\}$
  - Sentiment classification
- Sequence labeling: $h : \mathcal{V}^n \to \{0, \ldots, K\}^n$
  - Part-of-speech tagging
- Sequence generation: $h : \mathcal{V}_{\text{in}}^n \to \mathcal{V}_{\text{out}}^m$
  - Summarization: document to summary
  - In general: sequence to sequence

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?

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?
- 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?
- 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.

Autoregressive generation

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

Autoregressive generation

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 a sequence of classification problems:

1. Le Programme a ate mis en application $\rightarrow$ The

Autoregressive generation

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 a sequence of classification problems:

1. Le Programme a ate mis en application $\rightarrow$ The
2. Le Programme a ate mis en application, The $\rightarrow$ program

Autoregressive generation

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 a sequence of classification problems:

1. Le Programme a ate mis en application $\rightarrow$ The
2. Le Programme a ate mis en application, The $\rightarrow$ program
3. Le Programme a ate mis en application, The program $\rightarrow$ has

Autoregressive generation

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 a sequence of classification problems:

1. Le Programme a ate mis en application $\rightarrow$ The
2. Le Programme a ate mis en application, The $\rightarrow$ program
3. Le Programme a ate mis en application, The program $\rightarrow$ has
4. Le Programme a ate mis en application, The program has $\rightarrow$ been

Autoregressive generation

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 a sequence of classification problems:

1. Le Programme a ate mis en application $\rightarrow$ The
2. Le Programme a ate mis en application, The $\rightarrow$ program
3. Le Programme a ate mis en application, The program $\rightarrow$ has
4. Le Programme a ate mis en application, The program has $\rightarrow$ been
5. ...

Autoregressive generation

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 a sequence of classification problems:

1. Le Programme a ate mis en application $\rightarrow$ The
2. Le Programme a ate mis en application, The $\rightarrow$ program
3. Le Programme a ate mis en application, The program $\rightarrow$ has
4. Le Programme a ate mis en application, The program has $\rightarrow$ been
5. ...

Autoregressive generation

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 a sequence of classification problems:

1. Le Programme a ate mis en application $\rightarrow$ The
2. Le Programme a ate mis en application, The $\rightarrow$ program
3. Le Programme a ate mis en application, The program $\rightarrow$ has
4. Le Programme a ate mis en application, The program has $\rightarrow$ been
5. ...

We know how to solve each sequence classification problem!

The encoder-decoder architecture

We need a new module for autoregressive generation:

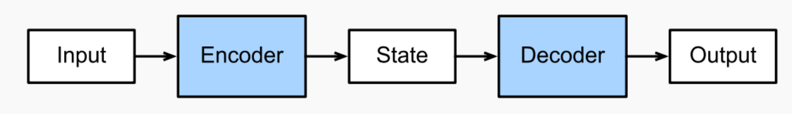


Figure: 10.6.1 from d2l.ai

The encoder-decoder architecture

We need a new module for autoregressive generation:

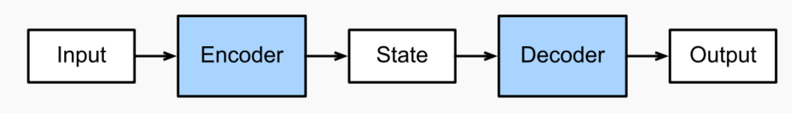


Figure: 10.6.1 from d2l.ai

- The **encoder** reads the input:

$$\text{Encoder}(x_1, \dots, x_n) = [h_1, \dots, h_n]$$

where $h_i \in \mathbb{R}^d$ are hidden states / embeddings.

The encoder-decoder architecture

We need a new module for autoregressive generation:

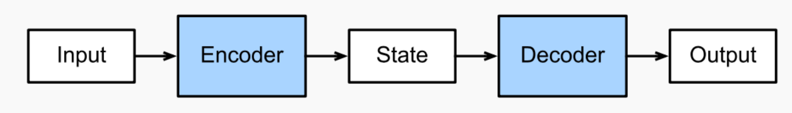


Figure: 10.6.1 from d2l.ai

- The **encoder** reads the input:

$$\text{Encoder}(x_1, \dots, x_n) = [h_1, \dots, h_n]$$

  where $h_i \in \mathbb{R}^d$ are hidden states / embeddings.

- The **decoder** writes the output:

$$\text{Decoder}(h_1, \dots, h_n) = [y_1, \dots, y_m]$$

  .

Autoregressive generative models

Generating sequences one token at a time from left to right

$$\text{Encoder}(x_1, \dots, x_n) = [h_1, \dots, h_n]$$

1. $\text{Decoder}([h_1, \dots, h_n]) \to y_1$
2. $\text{Decoder}([h_1, \dots, h_n], y_1) \to y_2$
3. $\text{Decoder}([h_1, \dots, h_n], y_1, y_2) \to y_3$
4. $\text{Decoder}([h_1, \dots, h_n], y_1, y_2, y_3) \to y_4$
5. ...

Autoregressive generative models

Is this the only way of modeling and generating text?

Autoregressive generative models

Is this the only way of modeling and generating text?

We want to learn $p(y \mid x)$

- Decompose the probability using **chain rule of probability**

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

- But we don't have to decompose it from left to right

Alternatives: reverse / right to left, parallel (non-autoregressive or diffusion models)

Table of Contents

RNN encoder-decoder model

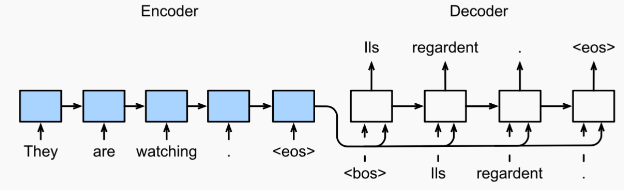


Figure: 10.7.1 from d2l.ai

- 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

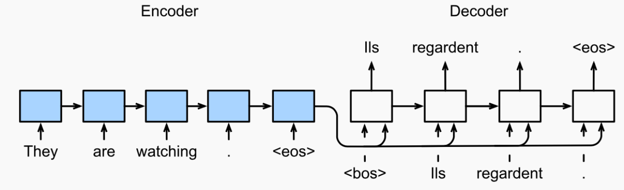


Figure: 10.7.1 from d2l.ai

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

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

- The **decoder** produce the output states recurrently and map them to distributions over the output vocabulary

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

Bi-directional RNN encoder

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

Bi-directional RNN encoder

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

We may want the hidden state to summarize both left and right context

Bi-directional RNN encoder

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

We may want the hidden state to summarize both left and right context

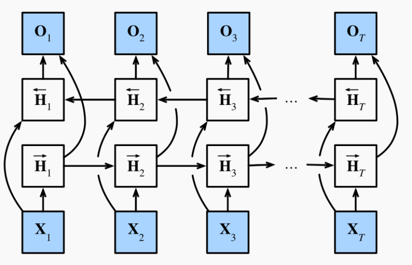


Figure: 10.4.1 from d2l.ai

- 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

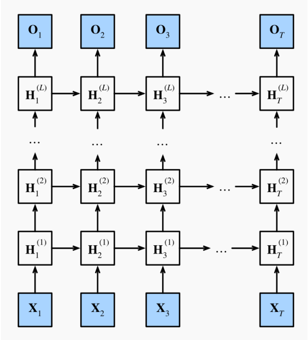


Figure: 10.3.1 from d2l.ai

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

Encoder-decoder attention: motivation

Recall that the context vector summarizes the input:

$$s_t = \mathrm{RNNDecoder}([y_{t-1}; c], s_{t-1})$$

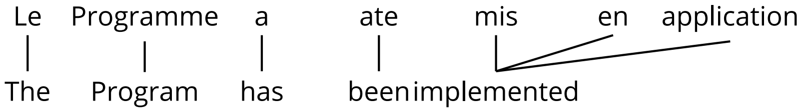
Should we use the same context vector for every decoding step?

Encoder-decoder attention: motivation

Recall that the context vector summarizes the input:

$$s_t = \mathrm{RNNDecoder}([y_{t-1}; c], s_{t-1})$$

Should we use the same context vector for every decoding step?



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

Encoder-decoder attention: motivation

Gradient vanishing for long distance dependence

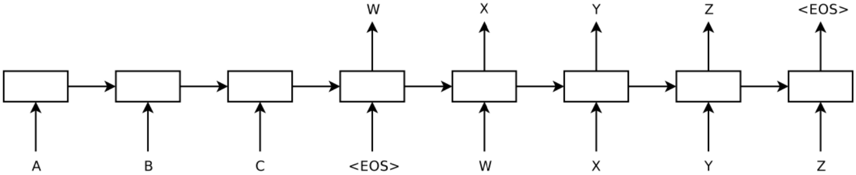


Figure: From Sequence to Sequence Learning with Neural Networks [Sutskever et al., 2014]

Encoder-decoder attention: motivation

Gradient vanishing for long distance dependence

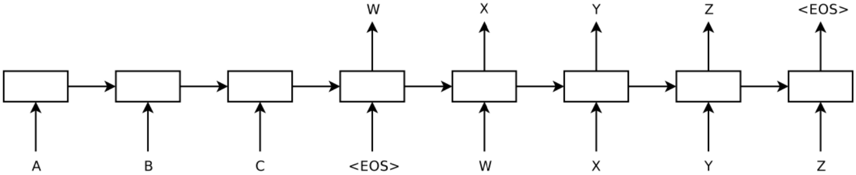


Figure: From Sequence to Sequence Learning with Neural Networks [Sutskever et al., 2014]

We may want gradient to flow more directly from input to output

Encoder-decoder attention: formalization

Recall that attention interacts pairs of tokens.

Decoder: Which input tokens are most relevant for generating the next output token?

Encoder-decoder attention: formalization

Recall that attention interacts pairs of tokens.

Decoder: Which input tokens are most relevant for generating the next output token?

- Query: decoder states s_{t-1}

Encoder-decoder attention: formalization

Recall that attention interacts pairs of tokens.

Decoder: Which input tokens are most relevant for generating the next output token?

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

Encoder-decoder attention: formalization

Recall that attention interacts pairs of tokens.

Decoder: Which input tokens are most relevant for generating the next output token?

- 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: formalization

Recall that attention interacts pairs of tokens.

Decoder: Which input tokens are most relevant for generating the next output token?

- 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: formalization

Recall that attention interacts pairs of tokens.

Decoder: Which input tokens are most relevant for generating the next output token?

- 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: $s_t = \mathrm{RNNDecoder}([y_{t-1}; c_t], s_{t-1})$

Encoder-decoder attention: formalization

Recall that attention interacts pairs of tokens.

Decoder: Which input tokens are most relevant for generating the next output token?

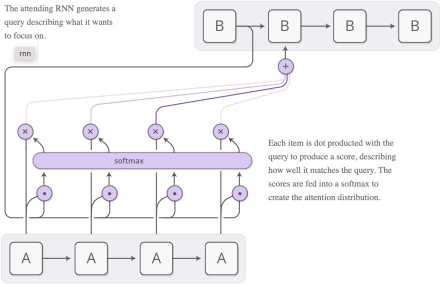
- Query: decoder states s_{t-1}
- Key: encoder states $h_1, \ldots, h_n$
- Value: encoder states $h_1, \ldots, h_n$
- Attention context: $c_t = \sum_{i=1}^{n} \alpha(s_{t-1}, h_i) h_i$
- Next state: $s_t = \mathrm{RNNDecoder}([y_{t-1}; c_t], s_{t-1})$
  - Dynamic context vector instead of a fixed one

Encoder-decoder attention: formalization

Recall that attention interacts pairs of tokens.

Decoder: Which input tokens are most relevant for generating the next output token?

- 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: $s_t = \mathrm{RNNDecoder}([y_{t-1}; c_t], s_{t-1})$
    - Dynamic context vector instead of a fixed one



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

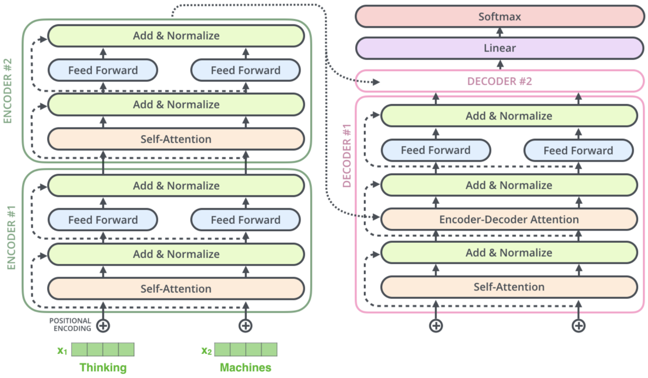


Figure: From illustrated transformer

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

Encoder-decoder attention in Transformer

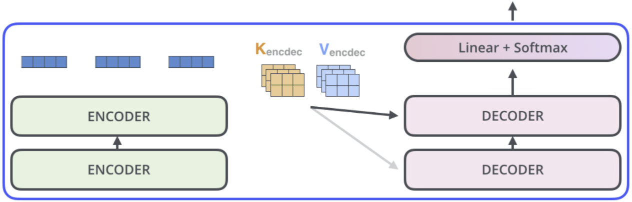


Figure: From illustrated transformer

$$\mathrm{TransformerEncoder}(x_1, \ldots, x_n) = [h_1, \ldots, h_n] = H_{\text{enc}} \quad (1)$$

$$K_{\text{encdec}} = H_{\text{enc}} W^K \quad (2)$$

$$V_{\text{encdec}} = H_{\text{enc}} W^V \quad (3)$$

(5)

Encoder-decoder attention in Transformer

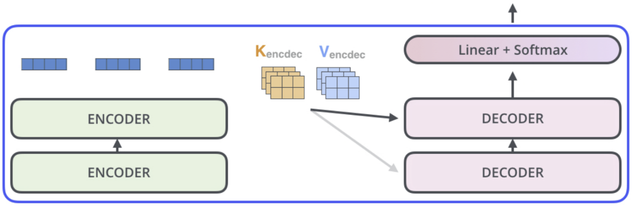


Figure: From illustrated transformer

$$\text{TransformerEncoder}(x_1, \ldots, x_n) = [h_1, \ldots, h_n] = H_{\text{enc}} \quad (1)$$

$$K_{\text{encdec}} = H_{\text{enc}} W^K \quad (2)$$

$$V_{\text{encdec}} = H_{\text{enc}} W^V \quad (3)$$

$$\text{DecoderSelfAttention}(y_1, \ldots, y_t) = [s_1, \ldots, s_t] \quad (4)$$

$$q_t = s_t W^Q \quad (5)$$

Table of Contents

Training

We are given a dataset $\mathcal{D} = \left\{ (x^{(i)}, y^{(i)}) \right\}_{i=1}^{N}$ of input and output sequences

Maximum likelihood estimation:

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

Training

We are given a dataset $\mathcal{D} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{N}$ of input and output sequences

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

We are given a dataset $\mathcal{D} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{N}$ of input and output sequences

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**)

Start and end symbols

Which one is more likely?

$$p(\text{The} \mid \text{Le Programme a ate mis en application})$$
$$p(\text{The program has been implemented} \mid \text{Le Programme a ate mis en application})$$

Start and end symbols

Which one is more likely?

$$p(\text{The} \mid \text{Le Programme a ate mis en application})$$
$$p(\text{The program has been implemented} \mid \text{Le Programme a ate mis en application})$$

Use sequence start and end symbols to model sequence length

- Le Programme a ate mis en application → <s> The ... </s>

Decoder attention masking for transformers

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 for transformers

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 $i = 1, \ldots, m$.

$$\mathrm{mask} = \begin{pmatrix} 0 & -\infty & -\infty & \cdots & -\infty \\ 0 & 0 & -\infty & \cdots & -\infty \\ 0 & 0 & 0 & \cdots & -\infty \\ \vdots & \vdots & \vdots & \ddots & -\infty \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}$$

Table of Contents

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:

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

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

Approximate search:

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

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

When to stop?

Approximate decoding: beam search

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

$$\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log p_\theta(y_i \mid y_{<i})$$

- At each step, we have a set of k partial hypotheses (prefixes)
- Use the autoregressive model, we can expand all hypotheses by one more token (how many hypotheses do we have now?)
- Evaluate the score of all hypotheses and keep the top k

Beam search example

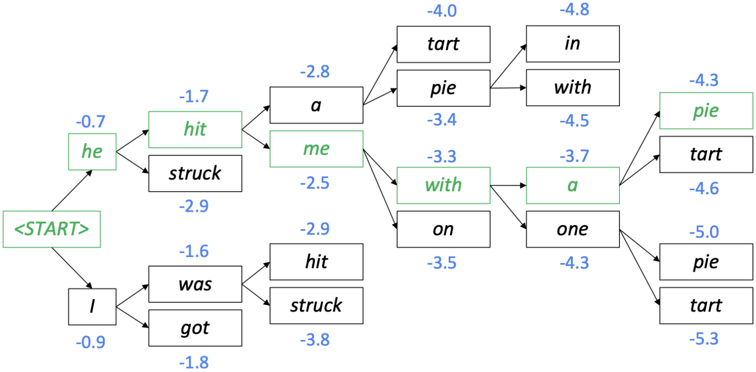


Figure: Figure from Chris Manning

Beam search example

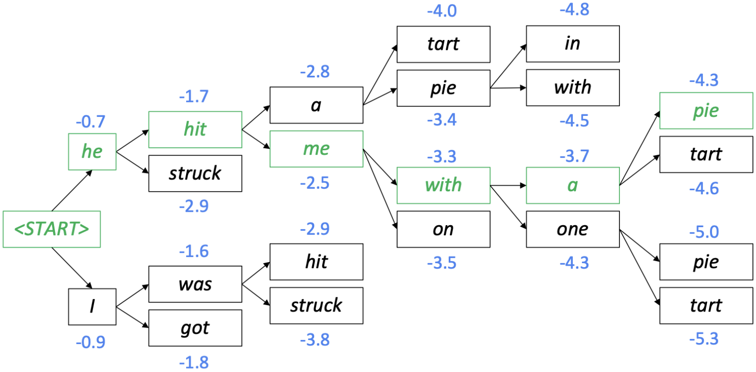
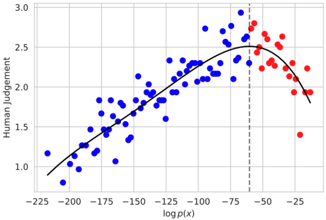


Figure: Figure from Chris Manning

Stop when all hypotheses in the beam has terminated or when hitting a limit of number of steps.

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 more evidence
- Training data contains repetition

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:

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

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?

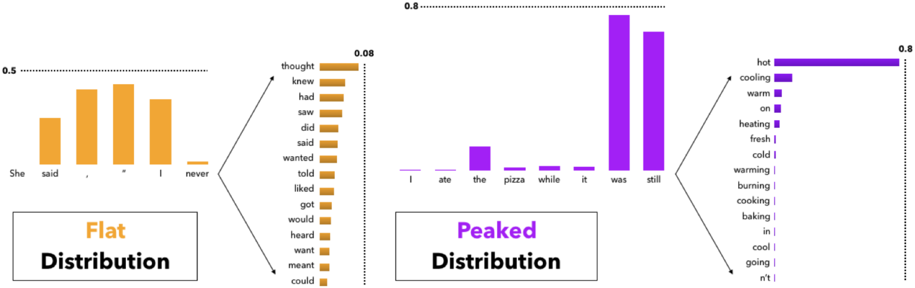


Figure: From the nucleus sampling paper by Holtzman et al., 2020

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 $> p$

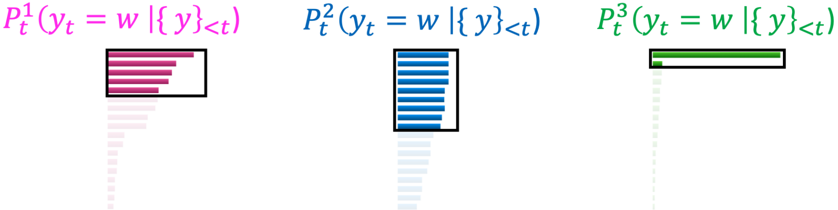
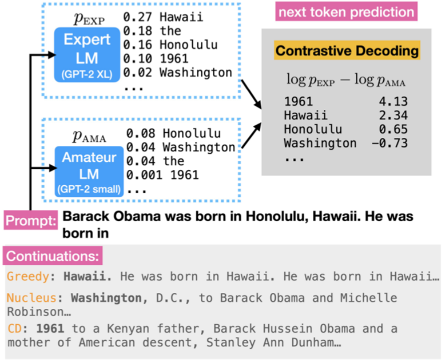


Figure: From Xiang Li's slides

Contrastive Decoding

- **Problem:** Greedy or beam search decoding can lead to repetitive or bland outputs.
- **Key Idea:** Such errors are more prominent in weaker/smaller models, so we can use a weaker model to penalize such errors [Li et al., 2023]



Contrastive Decoding

- Vanilla CD score:

$$\underbrace{\log p_{\text{strong}}(y_t \mid y_{<t})}_{\text{standard likelihood}} - \underbrace{\log p_{\text{weak}}(y_t \mid y_{<t})}_{\text{weak model penalty}},$$

- Remove (-inf score) implausible tokens $x_i \in \mathcal{V}$ where

$$p_{\text{strong}}(x_i \mid x_{<i}) < \alpha \max_{w \in \mathcal{V}} p_{\text{strong}}(w \mid x_{<i})$$

  Avoid low probability tokens that happen to have large contrast score

- Run beam search using CD score

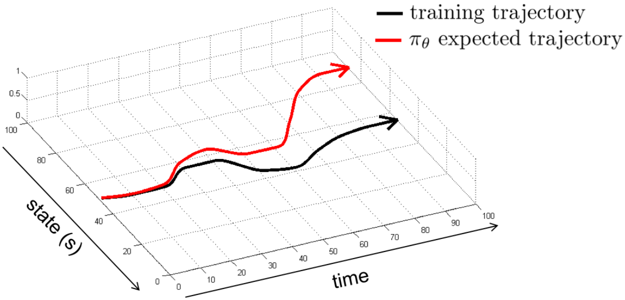
Decoding in practice

- 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
- Use top-k or top-p for open-ended generation, e.g. story generation, chit-chat dialogue
- 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