Neural Sequence Generation

He He

NEW YORK UNIVERSITY

February 14, 2023

Sequence generation

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

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

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**

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

- 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 $p(y_i \mid y_{<i}, x)$.

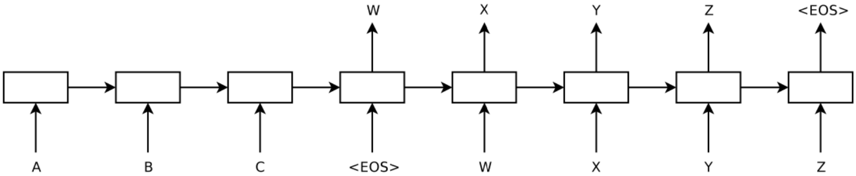


Figure: From Sequence to Sequence Learning with Neural Networks [Sutskever et al., 2014]

The encoder-decoder architecture

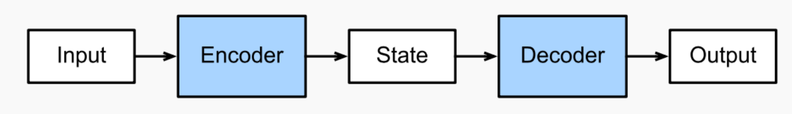


Figure: 10.6.1 from d2l.ai

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

The encoder-decoder architecture

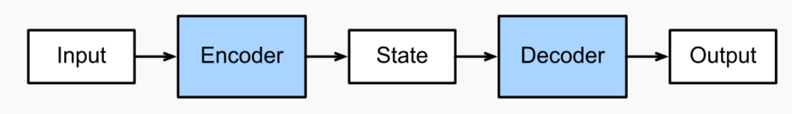


Figure: 10.6.1 from d2l.ai

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

- The **encoder** reads the input:

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

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

- The **decoder** writes the output:

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

.

RNN encoder-decoder model

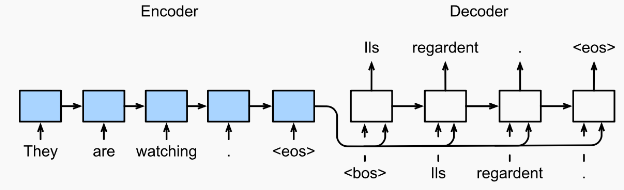


Figure: 10.7.1 from d2l.ai

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

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

- 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

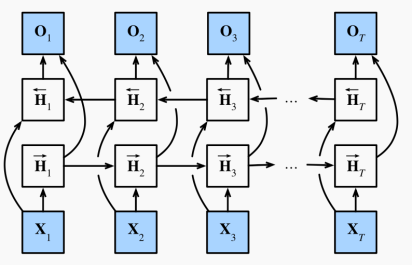


Figure: 10.4.1 from d2l.ai

- 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

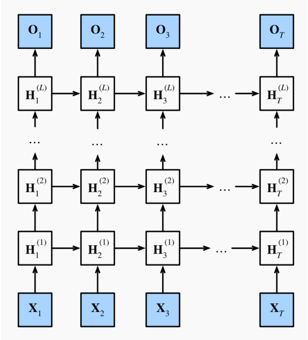
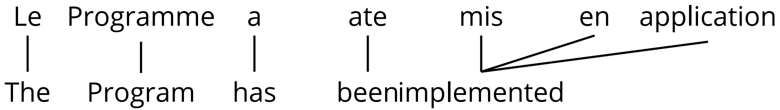


Figure: 10.3.1 from d2l.ai

- 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?



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?

Le Programme a ate mis en application

The Program has beenimplemented

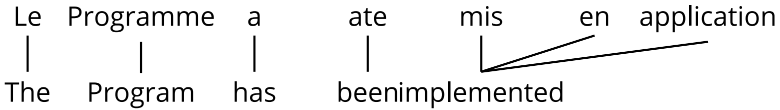
We may want to “look at” different parts of the input during decoding.

Think the database analogy:

- Query: decoder states s_{t-1}

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 $h_1, \ldots, h_n$

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

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 $h_1, \ldots, h_n$
- 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

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

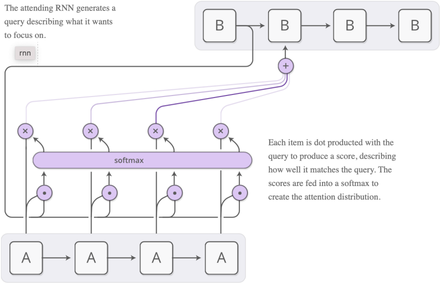
Le Programme a ate mis en application

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



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

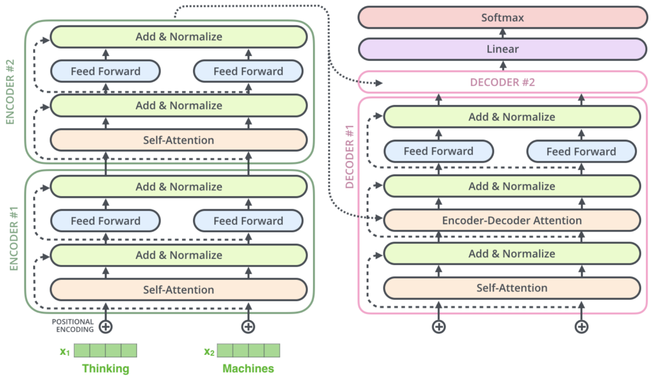


Figure: From illustrated transformer

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

Transformer encoder decoder model

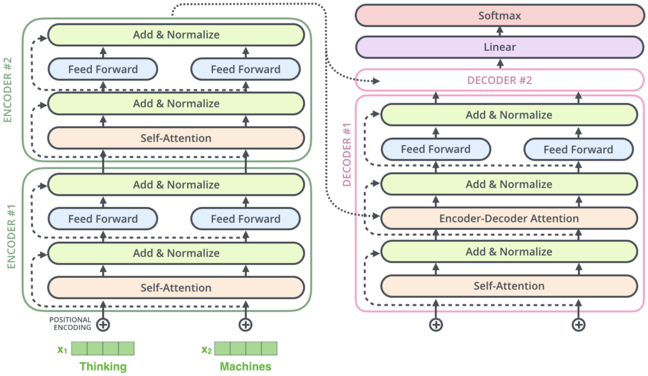


Figure: From illustrated transformer

- 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 (in two weeks)

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 should be 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 should be the prefix $y_{<j}$?

Option 1: whatever generated by the model

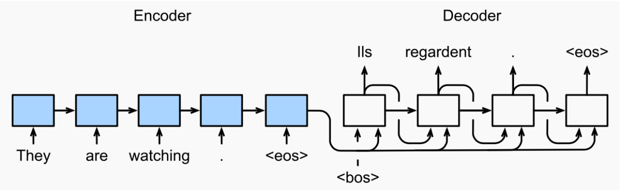


Figure: 10.7.1 from d2l.ai

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 should be the prefix $y_{<j}$?

Option 2: the groundtruth prefix (**teacher forcing**)

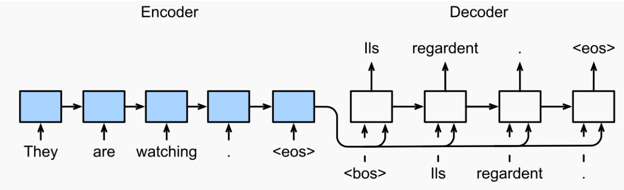


Figure: 10.7.3 from d2l.ai

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

How do we generate sequences given a trained model?

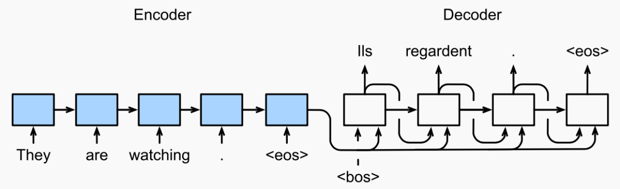


Figure: 10.7.1 from d2l.ai

The encoder-decoder model defines a probability distribution $p(y \mid x; \theta)$ over sequences.

Which one should we output?

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:

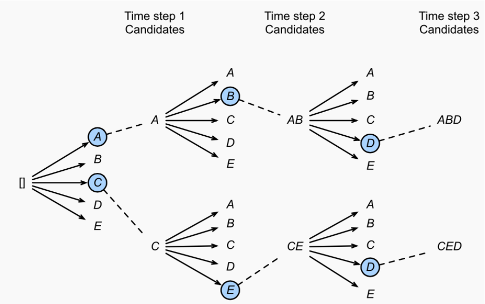
- **Greedy decoding**: return the most likely symbol at each step

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

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!

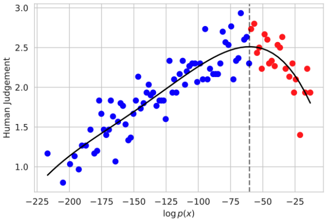


Figure: From the likelihood trap paper by Zhang et al., 2020

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..."
- Degraded generations with large beam size in MT

Sampling-based decoding

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

Sampling the next word sequentially:

- 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 the next word sequentially:

- 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.

Typically we modify the learned distrubtion p_θ before sampling the next word

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:

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

- 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 high likelihood 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

Truncated sampling

Another way to focus on high likelihood 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

Which k to choose?

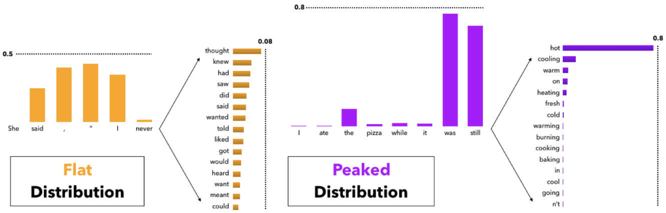


Figure: From the nucleus sampling paper by Holtzman et al., 2020

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 is larger than p

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