Natural Language Processing with Deep Learning

Lecture 10 — Decoder-only models and GPT

Prof. Dr. Ivan Habernal

January 16, 2025

www.trusthlt.org
Trustworthy Human Language Technologies Group (TrustHLT)
Ruhr University Bochum & Research Center Trustworthy Data Science and Security

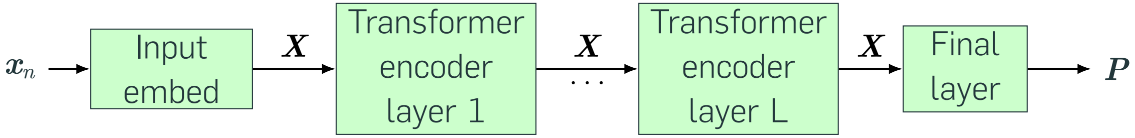
Motivation

We introduced BERT, a powerful transformer model for learning contextualized token embeddings

BERT can be used for

- text classification (one sequence, two concatenated sequences)
- sequence labeling (classify each token, e.g., NER, POS)

Transformer encoder (BERT)



For each input token, BERT produces contextualized word embeddings

Motivation

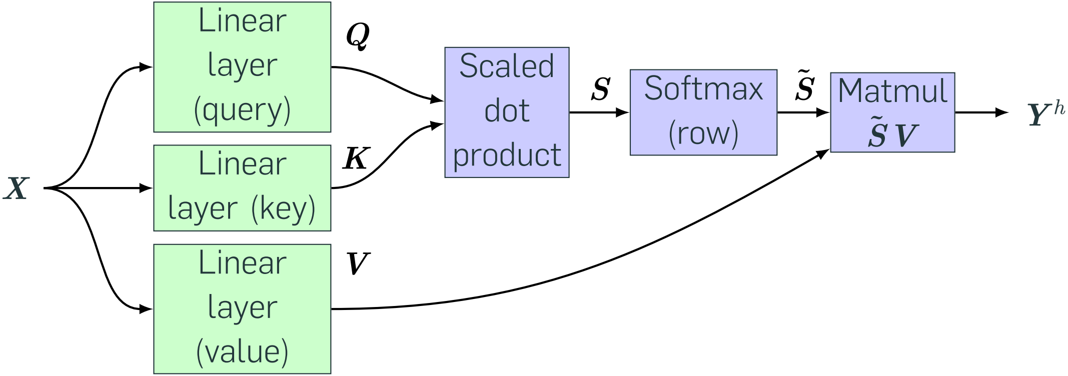
Although BERT is pre-trained with masked-language modeling, it is **not designed to generate text** by predicting the next token

Why?

We mask random tokens from the sequence and perform self-attention over past and future tokens

Can we use a transformer as a 'true' language model, aka. to conditionally generate text?

Recap: Single unmasked self-attention head (BERT)



Recap: Bidirectional / unmasked self-attention

Input: $\boldsymbol{X} \in \mathbb{R}^{\ell_\text{x} \times d_\text{x}}$, vector representations of the sequence of length ℓ_x

Output: $\tilde{\boldsymbol{V}} \in \mathbb{R}^{\ell_\text{x} \times d_\text{out}}$, updated vector representations of tokens in $\boldsymbol{X}$

Params $\boldsymbol{\mathcal{W}}_{\boldsymbol{qkv}}$: $\boldsymbol{W}_{\boldsymbol{q}}, \boldsymbol{W}_{\boldsymbol{k}} \in \mathbb{R}^{d_\text{x} \times d_\text{attn}}$, $\boldsymbol{b}_{\boldsymbol{q}}, \boldsymbol{b}_{\boldsymbol{k}} \in \mathbb{R}^{d_\text{attn}}$, $\boldsymbol{W}_{\boldsymbol{v}} \in \mathbb{R}^{d_\text{x} \times d_\text{out}}$, $\boldsymbol{b}_{\boldsymbol{v}} \in \mathbb{R}^{d_\text{out}}$

```
function Attention($\boldsymbol{X}; \boldsymbol{\mathcal{W}}_{\boldsymbol{qkv}}$)
    $\boldsymbol{Q} \leftarrow \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{q}} +_{\text{(rows)}} \boldsymbol{b}_{\boldsymbol{q}}$        ▷ Query $\in \mathbb{R}^{\ell_\text{x} \times d_\text{attn}}$
    $\boldsymbol{K} \leftarrow \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{k}} +_{\text{(rows)}} \boldsymbol{b}_{\boldsymbol{k}}$        ▷ Key $\in \mathbb{R}^{\ell_\text{x} \times d_\text{attn}}$
    $\boldsymbol{V} \leftarrow \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{v}} +_{\text{(rows)}} \boldsymbol{b}_{\boldsymbol{v}}$        ▷ Value $\in \mathbb{R}^{\ell_\text{x} \times d_\text{out}}$
    $\boldsymbol{S} \leftarrow \frac{1}{\sqrt{d_\text{attn}}}(\boldsymbol{Q}\boldsymbol{K}^\top)$        ▷ Scaled score $\in \mathbb{R}^{\ell_\text{x} \times \ell_\text{x}}$
    return $\tilde{\boldsymbol{V}} = \mathrm{softmax}_\text{row}(\boldsymbol{S})\,\boldsymbol{V}$
```

Recap: Basic single-query attention

Input: $\boldsymbol{e} \in \mathbb{R}^{d_{\text{in}}}$, vector representation of the current token

Input: $\boldsymbol{e}_t \in \mathbb{R}^{d_{\text{in}}}$, vector representations of the context tokens $t \in [T]$

Output: $\tilde{\boldsymbol{v}} \in \mathbb{R}^{d_{\text{out}}}$, vector representation of the token and context combined

Params: $\boldsymbol{W_q}, \boldsymbol{W_k} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{attn}}}$, $\boldsymbol{b_q}, \boldsymbol{b_k} \in \mathbb{R}^{d_{\text{attn}}}$, $\boldsymbol{W_v} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, $\boldsymbol{b_v} \in \mathbb{R}^{d_{\text{out}}}$

```
function Basic single-query attention
    $\boldsymbol{q} \leftarrow \boldsymbol{e}\boldsymbol{W_q} + \boldsymbol{b_q}$    ▷ Query linear projection
    for $t \in [T]$ do
        $\boldsymbol{k}_t \leftarrow \boldsymbol{e}_t \boldsymbol{W_k} + \boldsymbol{b_k}$    ▷ Key linear projection
        $\alpha_t = \frac{\exp(\boldsymbol{q} \cdot \boldsymbol{k_t} / \sqrt{d_{\text{attn}}})}{\sum_{u=1}^{T} \exp(\boldsymbol{q} \cdot \boldsymbol{k_u} / \sqrt{d_{\text{attn}}})}$    ▷ Softmax over scaled dot products, $\alpha_t \in (0, 1)$
        $\boldsymbol{v}_t \leftarrow \boldsymbol{e}_t \boldsymbol{W_v} + \boldsymbol{b_v}$    ▷ Value linear projection
    return $\tilde{\boldsymbol{v}} = \sum_{t=1}^{T} \alpha_t \boldsymbol{v}_t$
```

Example: Basic single-query unmasked attention

We are at position 2, our query $\boldsymbol{q} = (11, 12)$ and keys

$\boldsymbol{k_1} = (1, 2)$ $\boldsymbol{k_2} = (4, 5)$ $\boldsymbol{k_3} = (7, 8)$

$$\boldsymbol{q} = \begin{pmatrix} 11 & 12 \end{pmatrix} \quad \boldsymbol{K}^\top = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \end{pmatrix}$$

Dot products:

$$\boldsymbol{q} \cdot \boldsymbol{k_1} = (11, 12) \cdot (1, 2) = 11 + 24 = 35$$
$$\boldsymbol{q} \cdot \boldsymbol{k_2} = (11, 12) \cdot (4, 5) = 44 + 60 = 104$$
$$\boldsymbol{q} \cdot \boldsymbol{k_3} = (11, 12) \cdot (7, 8) = 77 + 96 = 173$$

Raw scores = $(35, 104, 173)$, after softmax (no scaling)

$\boldsymbol{\alpha} = (0.000\ldots, 0.000\ldots, 0.999\ldots)$

Example: Basic single-query unmasked attention

From previous slide

We are at position 2, our query $\boldsymbol{q} = (11, 12)$ and keys

$\boldsymbol{k}_1 = (1, 2) \qquad \boldsymbol{k}_2 = (4, 5) \qquad \boldsymbol{k}_3 = (7, 8)$

Raw scores = $(35, 104, 173)$, after softmax (no scaling)

$$\boldsymbol{\alpha} = (0.000\ldots, 0.000\ldots, 0.999\ldots)$$

Value at position 3 highest weight

- We are currently at position 2
- Position 3 is **in the future**

We want to attend only to previous tokens (4 token example)

At position 1 we should not attend to token 2, 3, and 4

At position 2 we should not attend to token 3 and 4

At position 3 we should not attend to token 4

At position 4 we can attend to all of them

$$\text{Raw associations} = \begin{pmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{pmatrix}$$

We want to assign zero probability (using softmax) to "future" tokens

We want to attend only to previous tokens

$$\text{Raw associations } = \begin{pmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{pmatrix}$$

Assign zero probability (using softmax) to "future" tokens

```
for $t \in [T]$ do
    $\boldsymbol{k_t} \leftarrow \ldots$
    $\alpha_t = \frac{\exp(\boldsymbol{q} \cdot \boldsymbol{k_t})}{\sum_{u=1}^{t} \exp(\boldsymbol{q} \cdot \boldsymbol{k_u})}$        ▷ Only until $t$
    for $i \in (t + 1, T)$ do
        $\alpha_i = 0$                                                        ▷ Zero-out rest
    $\boldsymbol{v_t} \leftarrow \ldots$
return $\tilde{\boldsymbol{v}} = \sum_{t=1}^{T} \alpha_t \boldsymbol{v}_t$
```

Avoid for-loops! How to vectorize this operation?

For each row $\boldsymbol{s}$ from the raw associations

```
$\alpha_t = \frac{\exp(\boldsymbol{s}_t)}{\sum_{u=1}^{t} \exp(\boldsymbol{s}_u)}$
for $i \in (t+1, T)$ do
    $\alpha_i = 0$
```

Replace input from $t+1$ onwards with $-\infty$

$$\text{Raw associations "masked"} = \begin{pmatrix} 11 & -\infty & -\infty & -\infty \\ 21 & 22 & -\infty & -\infty \\ 31 & 32 & 33 & -\infty \\ 41 & 42 & 43 & 44 \end{pmatrix}$$

Assigns zero probability (using softmax) to "future" tokens

Uni-directional masking for self-attention

For $t_z, t_x \in [\ell_x]$

$$\text{mask}[t_x, t_z] = \begin{cases} 1 & \text{if } t_z \leq t_x \\ 0 & \text{otherwise} \end{cases}$$

Example for $\ell_x = 4$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Creating the mask and indexing tensor by this mask very easy in pytorch

Left-to-right masked self-attention

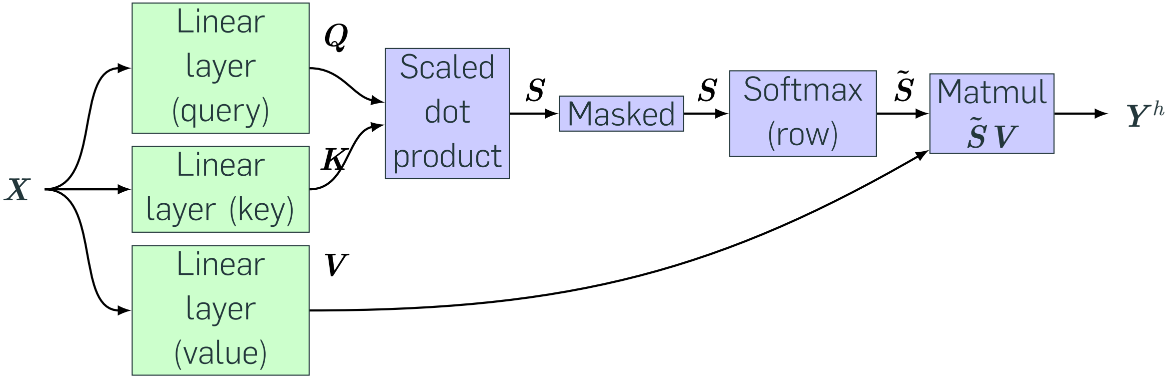
Input: $\boldsymbol{X} \in \mathbb{R}^{\ell_\mathsf{x} \times d_\mathsf{x}}$, vector representations of the sequence of length ℓ_x

Output: $\tilde{\boldsymbol{V}} \in \mathbb{R}^{\ell_\mathsf{x} \times d_\mathsf{out}}$, updated vector representations of tokens in $\boldsymbol{X}$

Params $\boldsymbol{\mathcal{W}_{qkv}}$: $\boldsymbol{W_q}, \boldsymbol{W_k} \in \mathbb{R}^{d_\mathsf{x} \times d_\mathsf{attn}}$, $\boldsymbol{b_q}, \boldsymbol{b_k} \in \mathbb{R}^{d_\mathsf{attn}}$, $\boldsymbol{W_v} \in \mathbb{R}^{d_\mathsf{x} \times d_\mathsf{out}}$, $\boldsymbol{b_v} \in \mathbb{R}^{d_\mathsf{out}}$

```
function Attention($\boldsymbol{X}; \boldsymbol{\mathcal{W}_{qkv}}$)
    $\boldsymbol{Q} \leftarrow \boldsymbol{X}\boldsymbol{W_q} +_{\text{(rows)}} \boldsymbol{b_q}$        ▷ Query $\in \mathbb{R}^{\ell_\mathsf{x} \times d_\mathsf{attn}}$
    $\boldsymbol{K} \leftarrow \boldsymbol{X}\boldsymbol{W_k} +_{\text{(rows)}} \boldsymbol{b_k}$        ▷ Key $\in \mathbb{R}^{\ell_\mathsf{x} \times d_\mathsf{attn}}$
    $\boldsymbol{V} \leftarrow \boldsymbol{X}\boldsymbol{W_v} +_{\text{(rows)}} \boldsymbol{b_v}$        ▷ Value $\in \mathbb{R}^{\ell_\mathsf{x} \times d_\mathsf{out}}$
    $\boldsymbol{S} \leftarrow \frac{1}{\sqrt{d_\mathsf{attn}}}(\boldsymbol{Q}\boldsymbol{K}^\top)$        ▷ Scaled score $\in \mathbb{R}^{\ell_\mathsf{x} \times \ell_\mathsf{x}}$
    for all $t_z, t_x \in [T]$ do
        if $\neg\,\mathrm{mask}[t_x, t_z]$ then $\boldsymbol{S}[t_x, t_z] \leftarrow -\infty$        ▷ Causal masking
    return $\tilde{\boldsymbol{V}} = \mathrm{softmax}_{\text{row}}(\boldsymbol{S})\,\boldsymbol{V}$
```

Single masked self-attention head (GPT)



Left-to-right masked self-attention

$$\tilde{\boldsymbol{V}} = \mathrm{softmax}_{\mathsf{row}}(\boldsymbol{S})\,\boldsymbol{V}$$

The output $\tilde{\boldsymbol{V}}[1:t,:]$ only depends on $\boldsymbol{X}[1:t,:]$, so it can be used to “predict” $\boldsymbol{X}[t+1,:]$

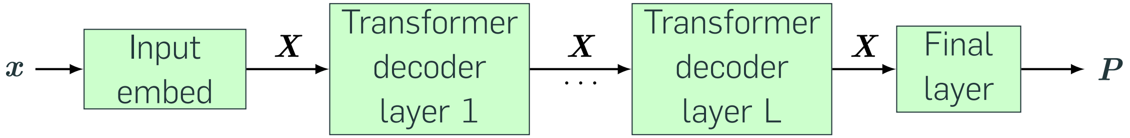
GPT (decoding-only transformer, forward pass)

```
function DTransformer($\boldsymbol{x}; \boldsymbol{\mathcal{W}}$)
    . . .
```

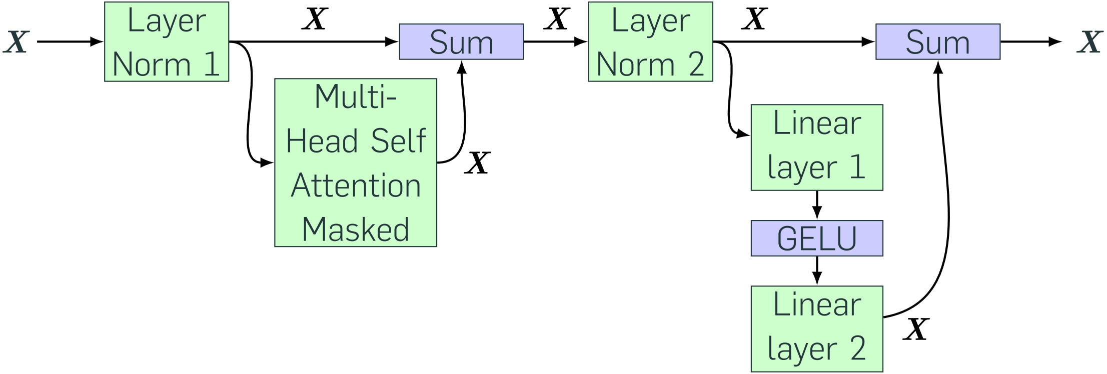
Input: — $\boldsymbol{x} \in V^*$, a sequence of token IDs, $\boldsymbol{\mathcal{W}}$ — all trainable parameters

Output: $\boldsymbol{P} \in (0, 1)^{\ell_\mathsf{x} \times N_\mathsf{V}}$, where each row of $\boldsymbol{P}$ is a distribution over the vocabulary conditioned on previous tokens $\hat{\boldsymbol{P}}(x[t+1] | \boldsymbol{x}[1:t])$

Transformer decoder (GPT)



GPT (decoding-only transformer, transformer layer)



GPT (decoding-only transformer, forward pass)

```
function DTransformer($\boldsymbol{x}; \boldsymbol{\mathcal{W}}$)
    $\ell \leftarrow$ length($\boldsymbol{x}$)
    for $t \in [\ell]$ : $\boldsymbol{e}_t \leftarrow \boldsymbol{W_e}[x[t], :] + \boldsymbol{W_p}[t, :]$        ▷ Token emb. + positional emb.
    $\boldsymbol{X} \leftarrow$ Stack row-wise$[\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots \boldsymbol{e}_\ell]$
    for $l = 1, 2, \ldots, L$ do
        $\boldsymbol{X} \leftarrow$ LayerNormPerRow$(\boldsymbol{X} | \boldsymbol{\gamma}_l^{\boldsymbol{1}}, \boldsymbol{\beta}_l^{\boldsymbol{1}})$        ▷ Normalization first
        $\boldsymbol{X} \leftarrow \boldsymbol{X} +$ MHAttentionMask$(\boldsymbol{X} | \boldsymbol{\mathcal{W}}_l)$        ▷ Just added masking
        $\boldsymbol{X} \leftarrow$ LayerNormPerRow$(\boldsymbol{X} | \boldsymbol{\gamma}_l^{\boldsymbol{2}}, \boldsymbol{\beta}_l^{\boldsymbol{2}})$
        $\boldsymbol{X} \leftarrow \boldsymbol{X} + \left(\text{GELU}(\boldsymbol{X}\boldsymbol{W}_l^{\text{mlp1}} +_{\text{(row)}} \boldsymbol{b}_l^{\text{mlp1}})\, \boldsymbol{W}_l^{\text{mlp2}} +_{\text{(row)}} \boldsymbol{b}_l^{\text{mlp2}}\right)$        ▷ MLP
    $\boldsymbol{X} \leftarrow$ LayerNormPerRow$(\boldsymbol{X} | \boldsymbol{\gamma}_l, \boldsymbol{\beta}_l)$
    return $\boldsymbol{P} = \mathrm{softmax}(\boldsymbol{X}\boldsymbol{W_u})$        ▷ Project to vocab., probabilities
```

GPT (decoding-only transformer, forward pass)

Differences from BERT forward-pass:

Switched the ordering of layer normalization (line 6 and 8)

No final layer projection

Attention with left-to-right masking (line 7)

Training

1 Training

2 Decoder model prompting

3 Evolution of GPT

Decoder-Transfomer: Training on next token prediction

```
function DTraining($\{\boldsymbol{x}_n\}_{n=1}^{N_\text{data}}$ seqs, $\boldsymbol{\theta}$ init. params, $N_\text{epochs}$, $\eta$)
    for $i \in [N_\text{epochs}]$ do
        for $n \in [N_\text{data}]$ do
            $\ell \leftarrow \text{length}(\boldsymbol{x}_n)$
            $\boldsymbol{P_\theta} \leftarrow \text{DTransformer}(\boldsymbol{x}_n | \boldsymbol{\theta})$
            $\text{loss}_{\boldsymbol{\theta}} \leftarrow -\sum_{t=1}^{\ell-1} \log \boldsymbol{P_\theta}[t, \boldsymbol{x}_n[t+1]]$
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \nabla \text{loss}_{\boldsymbol{\theta}}$
    return $\boldsymbol{\theta}$
```

Explaining line 6 (negative log likelihood)

$$\begin{pmatrix}\text{The} & \text{cat} & \text{sat}\end{pmatrix} \to \boldsymbol{x}_n = \begin{pmatrix}21 & 11987 & 5438\end{pmatrix}$$

$$\boldsymbol{P_\theta} \leftarrow \text{DTransformer}(\boldsymbol{x}_n | \boldsymbol{\theta})$$

$$\boldsymbol{P_\theta} = \begin{pmatrix} 0.001 & 0.0007 & \dots & 0.0003 \\ 0.0013 & 0.0065 & \dots & 0.0001 \\ 0.079 & 0.015 & \dots & 0.0001 \end{pmatrix}$$

$\boldsymbol{P_\theta} \in (0,1)^{\ell_\mathsf{x} \times N_\mathsf{V}}$, where each row of $\boldsymbol{P}$ is a distribution over the vocabulary

Explaining line 6 (negative log likelihood), $t = 1$

$$\boldsymbol{x}_n = (21, 11987, 5438) \; \boldsymbol{P_\theta} = \begin{pmatrix} 0.001 & \dots & 0.0041_{11987} & \dots 0.0003 \\ \vdots & & & \end{pmatrix}$$

For $t = 1$, the model should learn to predict "cat" (idx 11987)

Gold: $\boldsymbol{y} = (0, 0, \dots, 1_{11987}, \dots, 0) \in \mathbb{R}^{N_\text{V}}$

Pred: $\hat{\boldsymbol{y}} = \boldsymbol{P_\theta}[1, :] = (0.001, \dots, 0.0041_{11987}, \dots 0.0003) \in \mathbb{R}^{N_\text{V}}$

Categorical cross entropy loss (Lec. 4)

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) := - \sum_{k=1}^{K} \boldsymbol{y}_{[k]} \log \left(\hat{\boldsymbol{y}}_{[k]} \right)$$

$$= -1 \cdot \log(\hat{\boldsymbol{y}}[11987]) = - \log(\boldsymbol{P_\theta}[1, 11987])$$

$$= - \log(\boldsymbol{P_\theta}[1, \boldsymbol{x}_n[1 + 1]]) = - \log(\boldsymbol{P_\theta}[t, \boldsymbol{x}_n[t + 1]])$$

Decoder model prompting

1 Training

2 Decoder model prompting

3 Evolution of GPT

Decoding

Input: $\boldsymbol{x} \in V^*$, a prompt (sequence of token IDs)

Output: $\boldsymbol{y} \in V^*$, continuation

```
function DInference($\boldsymbol{x}, \boldsymbol{\theta}, \ell_{\text{gen}}, \tau$)
    $\ell \leftarrow$ length($\boldsymbol{x}$)
    for $i \in [\ell_{\text{gen}}]$ do
        $\boldsymbol{P} \leftarrow$ DTransformer($\boldsymbol{x}|\boldsymbol{\theta}$)
        $\boldsymbol{p} \leftarrow \boldsymbol{P}[\ell + i - 1, :]$
        sample token $y$ from $\boldsymbol{q} \propto \boldsymbol{p}^{(1/\tau)}$
        $\boldsymbol{x} \leftarrow [\boldsymbol{x}, y]$
    return $\boldsymbol{y} = \boldsymbol{x}[\ell + 1 : \ell + \ell_{\text{gen}}]$
```

Evolution of GPT

1 Training

2 Decoder model prompting

3 Evolution of GPT

Towards GPT-1

Decoder part of the Transformer Encoder-Decoder model for MT (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, L. Kaiser, and Polosukhin, 2017)

Dropping encoder and using only decoder that consumes input and produces output trained as a standard language model for writing Wikipedia pages as summarization task (Liu, Saleh, Pot, Goodrich, Sepassi, Ł. Kaiser, and Shazeer, 2018)

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). **“Attention Is All You Need”.** In: *Advances in Neural Information Processing Systems 30.* Long Beach, CA, USA: Curran Associates, Inc., pp. 5998–6008

P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, Ł. Kaiser, and N. Shazeer (2018). **“Generating Wikipedia by Summarizing Long Sequences”.** In: *Proceedings of the 6th International Conference on Learning Representations.* Vancouver, BC, Canada

GPT-1

GPT-1 (Radford, Narasimhan, Salimans, and Sutskever, 2018) adapted decoder only transformer

- pre-training as LM
- fine-tuning with an extra final layer for the given task
- pre-trained on BooksCorpus (7k unique unpublished books)
- 12 decoder layers, 12 attention heads, 768 embedding size

"improving the state of the art on 9 of the 12 datasets we study"

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever (2018). **Improving Language Understanding by Generative Pre-Training.** Technical report. OpenAI

GPT-2

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever (2019). **Language Models are Unsupervised Multitask Learners.** Technical report. OpenAI

Larger GPT-1

- pre-training as LM
- pre-trained on custom web scrape (all outbounds links from Reddit with at least 3 karma points, for quality reasons), 8 million documents total
- 48 decoder layers, 1600 embedding size (1.542 billion params)

Representing inputs, prompting, etc. — next lectures

License and credits

Credits

Ivan Habernal

Content from ACL Anthology papers licensed under CC-BY
`https://www.aclweb.org/anthology`