AMATH 840: Advanced Numerical Methods for Computational and Data Science

Winter 2024

Part 2: Neural Networks
2.2: Transformers - Function Representations

Prepared by Yanming Kang and Giang Tran

Winter 2024

Outline

Natural Language Processing (NLP)

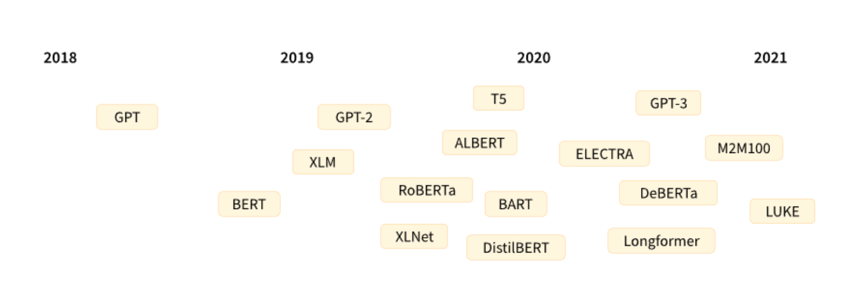
Examples of NLP tasks:

- Classifying whole sentences: sentiment analysis, email spam filter, grammar check, sentences' correlations.
- Classifying each word in a sentence: noun/verb/adjective, named entity recognition(person/location/organization...)
- Text generation: Completing a prompt, filling in the blanks in a text.
- Question-Answering (extractive summarization).
- Generating a new sentence from an input text (seq2seq): Translation, abstractive summarization.

[1]

[1] https://huggingface.co/course/

Transformers



- Pretrained language models (backbone models): Trained on large amounts of raw text, the models' sizes are big.
- Transfomer models are self-supervised learning: the objective is automatically computed from the inputs of the model. No need of labeled data.

2

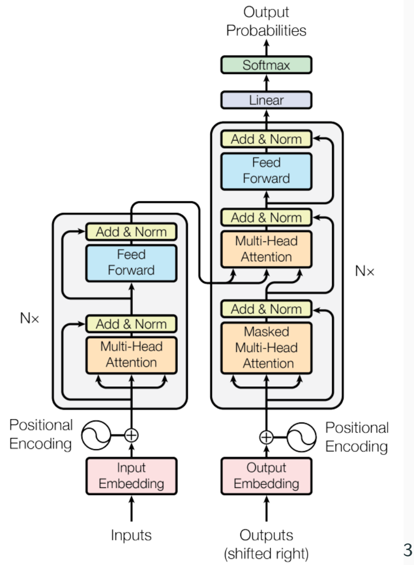
[2] `https://huggingface.co/course/`

General Transformer Architecture

- Basic blocks of a transfomer model:
  - Encoder: Input $\xrightarrow{\textit{Encoder}}$ A representation (features) of the input.
  - Decoder: Encoder's representation + other inputs $\xrightarrow{\textit{Decoder}}$ Generate a target sequence.
- Encoder-only models: Good for task that require understanding of the input, such as sentence classification, named entity recognition, QA. Examples: BERT, RoBERTa, DeBERTa.
- Decoder-only models: Good for generative tasks such as text generation. Examples: GPT, GPT-2,3,4, Transformer XL.
- Encoder-decoder model: Good for generative tasks that require an input, such as translation or summarization. Examples: BART, mBART, Marian, T5.

Outline

Vanilla Transformer Model



[3]

[3] "Attention is all you need", `https://arxiv.org/abs/1706.03762`

Vanilla Transformer Model (cont'd)

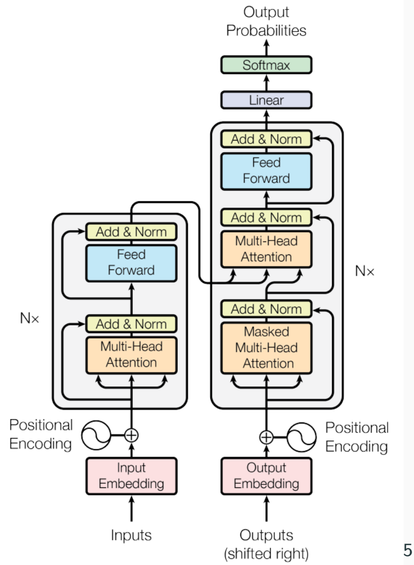
- The original Transformer was designed for translation.
- In the translation task, during training, the encoder receives inputs (sentences) in a certain language while the decoder receives the same sentence in another language.
- Inputs to Encoder: $\begin{bmatrix} w_1 & w_2 & \dots & w_n \end{bmatrix} \in \mathbb{R}^{n \times v}$, where n is the sequence length and v is vocab size. Each w_t is a word (or character) in the sequence.
- $w_t \in \mathbb{R}^v \xrightarrow{\text{Input Embedding}} e(w_t) \in \mathbb{R}^{d_{model}}$, for $t = 1, \dots, n$.
- Positional Encoding: Re-represent the values of a word and its position in a sentence. For example[4],

$$\begin{aligned} e^{new}(w_t) &= e(w_t) + \vec{p}_t \\ &= e(w_t) + \left[\sin(\omega_1 t), \cos(\omega_1 t), \cdots, \sin(\omega_{d/2} t), \cos(\omega_{d/2} t)\right]^T, \end{aligned}$$

where $\omega_k = 10^{-10k/d}$, $d = d_{model}$.

[4] https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Vanilla Transformer Model(cont'd)



[5]

[5] Attention is all you need, https://arxiv.org/abs/1706.03762

Vanilla Transformer Model - Encoder Part

- Each encoder layer has two residual blocks:
  1. A multi-head self-attention
  2. A feed-forward NN

- In the encoder, the attention layers can use all the words in a sentence (since the translation of a given word can be dependent on what is after as well as before it in the sentence).[6]

- Each followed by a layernorm (normalize each sample such that the elements in the sample have zero mean and unit variance).

- In practice, a dropout layer is added between additions and layernorms.

[6] https://huggingface.co/course/chapter1/4?fw=pt

Vanilla Transformer Model - Decoder Part

- Each decoder layer has three residual blocks:
  1. A causally masked multi-head self-attention (later)
  2. A cross attention where the keys and values come from the output of the encoder $\rightarrow$ Access the whole input sentence to best predict the current word.
  3. A feed-forward NN

- The decoder works sequentially and can only pay attention to the words in the sentence that it has already translated. [7]

- Masked multi-head attention: the upper triangular part (excluding the diagonal) of QK^T is set to $-\infty$ to ensure that the result at every position does not depend on subsequent values in V.

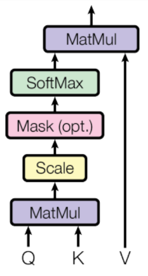
- Each followed by a layernorm.

- In practice, a dropout layer is added between additions and layernorms.

[7] https://huggingface.co/course/chapter1/4?fw=pt

Attention Layers

- Key feature of Transformer models is the attention layers.
- Roles of attention layers:
  - Pay specific attention to certain words in the sentence when dealing with the representation of each word.
  - The meaning of a word is affected by the context, which can be any word (or words) before or after the word being studied.



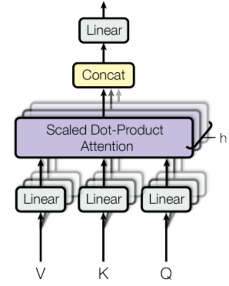


Figure 1: Scaled dot-product attention and Multihead attention.

Attention Layers (cont'd)

- The scaled dot-product attention is defined as

$$\texttt{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \texttt{softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right)\mathbf{V} \in \mathbb{R}^{m \times d_v},$$

  where $\mathbf{Q} \in \mathbb{R}^{m \times d_k}$, $\mathbf{K} \in \mathbb{R}^{n \times d_k}$ and $\mathbf{V} \in \mathbb{R}^{n \times d_v}$.

- Masked attention: set the upper-triangular part of $\mathbf{QK}^T$ to $-\infty$. After $\texttt{softmax}$, these entries become 0. This ensures autoregressiveness - output at time t only depends on inputs of $< t$.

- The multihead attention is given by:

$$\mathbf{A} = [\texttt{Attn}(\mathbf{XW}_Q^1, \mathbf{YW}_K^1, \mathbf{ZW}_V^1), \dots, \texttt{Attn}(\mathbf{XW}_Q^h, \mathbf{YW}_K^h, \mathbf{ZW}_V^h)]\mathbf{W}_O,$$

  where $\mathbf{X} \in \mathbb{R}^{q \times d_{\text{model}}}$; $\mathbf{Y}, \mathbf{Z} \in \mathbb{R}^{n \times d_{\text{model}}}$ are the inputs, and all $\mathbf{W}$'s are trainable parameters:

$$\mathbf{W}_Q^i, \mathbf{W}_K^i \in \mathbb{R}^{d_{\text{model}} \times d_k}, \mathbf{W}_V^i \in \mathbb{R}^{d_{\text{model}} \times d_v}, \mathbf{W}_O \in \mathbb{R}^{hd_v \times d_{\text{model}}}.$$

- The multihead self-attention: multihead attention with $\mathbf{X} = \mathbf{Y} = \mathbf{Z}$.

Scaled Dot-Product Attention

- Let $\mathbf{q}_1, \dots, \mathbf{q}_m \in \mathbb{R}^{d_k}$ be the rows of $\mathbf{Q}$;
  $\mathbf{k}_1, \dots, \mathbf{k}_n \in \mathbb{R}^{d_k}$ be the rows of $\mathbf{K}$;
  $\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^{d_v}$ be the rows of $\mathbf{V}$,
  and $\mathbf{S} = (s_{i,j}) := \mathtt{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)$.
- Then

$$\mathtt{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathtt{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

$$= \begin{pmatrix} \mathtt{softmax}\begin{pmatrix} \frac{\mathbf{k}_1^T\mathbf{q}_1}{\sqrt{d_k}} & \dots & \frac{\mathbf{k}_n^T\mathbf{q}_1}{\sqrt{d_k}} \end{pmatrix} \\ \mathtt{softmax}\begin{pmatrix} \frac{\mathbf{k}_1^T\mathbf{q}_2}{\sqrt{d_k}} & \dots & \frac{\mathbf{k}_n^T\mathbf{q}_2}{\sqrt{d_k}} \end{pmatrix} \\ \vdots \\ \mathtt{softmax}\begin{pmatrix} \frac{\mathbf{k}_1^T\mathbf{q}_m}{\sqrt{d_k}} & \dots & \frac{\mathbf{k}_n^T\mathbf{q}_m}{\sqrt{d_k}} \end{pmatrix} \end{pmatrix}\mathbf{V} = \begin{pmatrix} \left(\sum_{j=1}^{n} s_{1,j}\mathbf{v}_j\right)^T \\ \left(\sum_{j=1}^{n} s_{2,j}\mathbf{v}_j\right)^T \\ \vdots \\ \left(\sum_{j=1}^{n} s_{m,j}\mathbf{v}_j\right)^T \end{pmatrix}.$$

Scaled Dot-Product Attention (cont'd)

Comments:

- Notations: In these slides, all indices start from 1 and all vectors are column vectors of size $d \times 1$. Note that, in Pytorch, all indices starts from 0 and vectors are of size $1 \times d$, with suitable d.
- The `softmax` operator is applied to each row of $\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right)$.
- Each row of the scaled dot-product attention is a linear combination of rows of $\mathbf{V}$, where the weights (coefficients) are decided by the relations (similarity) between rows of $\mathbf{Q}$ and $\mathbf{K}$.
- Potential benefit: Since the entries of $\mathcal{S}$ are obtained by the dot products of every row of $\mathbf{Q}$ with every row of $\mathbf{K}$, if $\mathbf{Q} = \mathbf{XW}_Q$ and $\mathbf{K} = \mathbf{XW}_K$, then all words in $\mathbf{X}$ are paid attention to all other words in $\mathbf{X}$ $\Rightarrow$ Useful for language translation, for example.

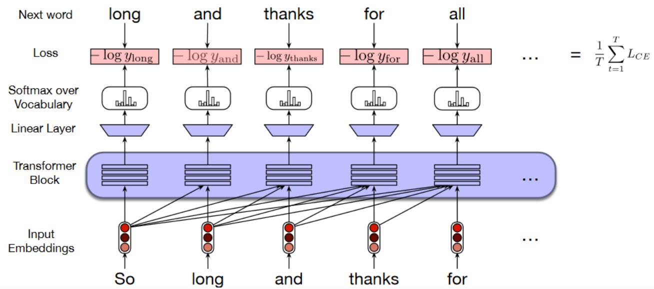
Outline

Language Models

- Two classes of LMs:
  1. Autoregressive (unidirectional) LM, e.g. GPT. Good for generation,
  2. Masked (bidirectional) LM, e.g. BERT. Good for classification.
- Pre-train + fine-tune regime
  - idea from computer vision
  - labeled datasets are small and few compared to size of models - overfitting, bad generalizing ability.
  - Pre-train in self-supervised way on very large unlabeled datasets. ("The Pile" 825GB)
  - Fine-tune on small task-specific labeled dataset for a small number of epochs. Better results than training solely on the small dataset.

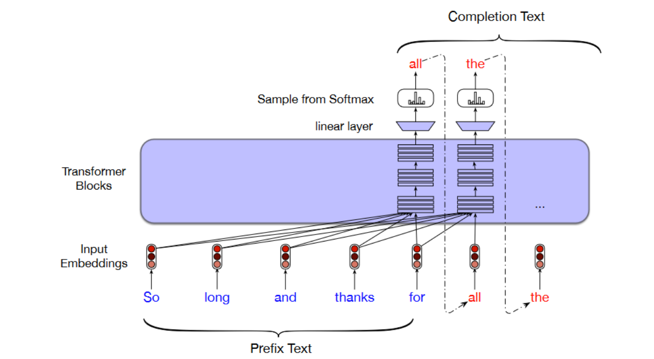
Autoregressive models: training

- "Decoder-only" models: the attention has a causal mask.
- Maximize the log-likelihood of each correct word x_t given previous ones $x_{<t}$.
- Teacher-Forced training: Because of the causal mask applied to the attention, the model can be trained in parallel in time. (Unlike RNN which has to be trained sequentially.)



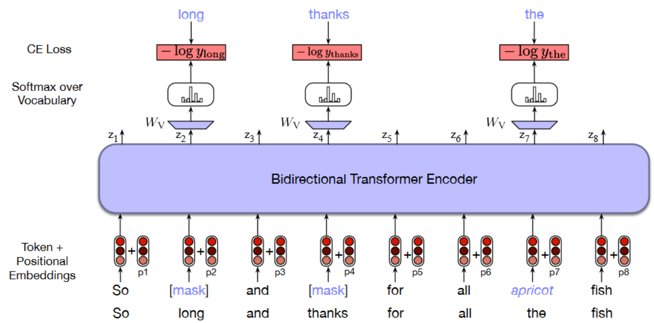
Autoregressive models: generation

- Generate outputs incrementally: Greedy or beam search.



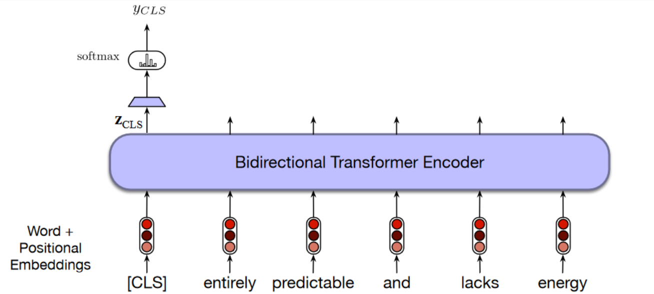
Masked Language Models: training

- Encoder-only models.
- Maximize the log-likelihood of masked words x_t given other words $x_{\neq t}$.



Masked Language Models: fine-tuning for sentence classification

- Use the output vector for the [CLS] token (seen in pre-training).
- Remove the language modeling head and add a classifier head.



Bias and Limitations of Transformer Models

- Pretrained models: Training a model from scratch on very large amounts of data$\rightarrow$ The training data may contain both good and bad data.
- The pretrained models could generate sexist, racist, or homophobic content - Gemini.
- Fine tuning the model on your data won't make this intrinsic bias disappear.

Outline

Self-Attention and Nonparametric Kernel Regression[8]

- Training set $\{\mathbf{k}_j, \mathbf{v}_j\}_{j=1}^{N}$, where $\mathbf{k}_j$ are the key vectors (training inputs), and $\mathbf{v}_j$ are the value vectors (training outputs).
- Nonparametric regression model: Learn a function f such that

$$\mathbf{v}_j = f(\mathbf{k}_j) + \varepsilon_j, \quad \forall j = 1, \ldots, N,$$

  where ε_j are independent noises with zero mean; $\mathbf{k}_j$ are i.i.d. samples from the distribution that admits $p(\mathbf{k})$ as density function.
- Nadaraya-Watson estimator: $\mathbb{E}[\mathbf{v}_j|\mathbf{k}_j] = f(\mathbf{k}_j)$, for all $j \in [n]$.

[8] https://arxiv.org/abs/2206.00206

Self-Attention and Nonparametric Kernel Regression(cont'd)

- Denote $p(\mathbf{v}, \mathbf{k})$ the joint density where the key and value vectors $\{\mathbf{k}_j, \mathbf{v}_j\}_{j=1}^{N}$ are i.i.d. samples from. We have

$$\mathbb{E}[\mathbf{v}|\mathbf{k}] = \int_{\mathbb{R}^d} \mathbf{v} \cdot p(\mathbf{v}|\mathbf{k}) d\mathbf{v} = \int \frac{v \cdot p(\mathbf{v}, \mathbf{k})}{p(\mathbf{k})} d\mathbf{v}$$

- Using isotropic Gaussian kernel with bandwidth σ to approximate $p(\mathbf{v}, \mathbf{k})$ and $p(\mathbf{k})$:

$$\hat{p}_\sigma(\mathbf{v}, \mathbf{k}) = \frac{1}{N} \sum_{j=1}^{N} \varphi_\sigma(\mathbf{v} - \mathbf{v}_j) \varphi_\sigma(\mathbf{k} - \mathbf{k}_j), \quad \hat{p}_\sigma(\mathbf{k}) = \frac{1}{N} \sum_{j=1}^{N} \varphi_\sigma(\mathbf{k} - \mathbf{k}_j),$$

where $\varphi_\sigma(\cdot)$ is the isotropic multivariate Gaussian density function with diagonal covariance matrix $\sigma^2 I_d$.

Self-Attention and Nonparametric Kernel Regression[9]

$$
\begin{aligned}
\widehat{f}_\sigma(\boldsymbol{k}) &= \int_{\mathbb{R}^D} \frac{\mathbf{v} \cdot \hat{p}_\sigma(\mathbf{v}, \boldsymbol{k})}{\hat{p}_\sigma(\boldsymbol{k})} d\mathbf{v} = \int_{\mathbb{R}^D} \frac{\mathbf{v} \cdot \sum_{j=1}^{N} \varphi_\sigma\left(\mathbf{v} - \mathbf{v}_j\right) \varphi_\sigma\left(\boldsymbol{k} - \boldsymbol{k}_j\right)}{\sum_{j=1}^{N} \varphi_\sigma\left(\boldsymbol{k} - \boldsymbol{k}_j\right)} d\mathbf{v} \\
&= \frac{\sum_{j=1}^{N} \phi_\sigma\left(\boldsymbol{k} - \boldsymbol{k}_j\right) \int \mathbf{v} \cdot \varphi_\sigma\left(\mathbf{v} - \mathbf{v}_j\right) d\mathbf{v}}{\sum_{j=1}^{N} \varphi_\sigma\left(\boldsymbol{k} - \boldsymbol{k}_j\right)} = \frac{\sum_{j=1}^{N} v_j \varphi_\sigma\left(\boldsymbol{k} - \boldsymbol{k}_j\right)}{\sum_{j=1}^{N} \varphi_\sigma\left(\boldsymbol{k} - \boldsymbol{k}_j\right)}
\end{aligned}
$$

[9] https://arxiv.org/abs/2206.00206

Self-Attention and Nonparametric Kernel Regression[10]

Connection between Self-Attention and nonparametric regression: By plugging the query vectors $\boldsymbol{q}_i$ into the function $\widehat{f}_\sigma$, we obtain that

$$\begin{aligned}\widehat{f}_\sigma\left(\boldsymbol{q}_i\right) &= \frac{\sum_j^N \mathbf{v}_j \exp\left(-\left\|\boldsymbol{q}_i - \boldsymbol{k}_j\right\|^2 / 2\sigma^2\right)}{\sum_j^N \exp\left(-\left\|\boldsymbol{q}_i - \boldsymbol{k}_j\right\|^2 / 2\sigma^2\right)} \\ &= \frac{\sum_j^N \mathbf{v}_j \exp\left[-\left(\left\|\boldsymbol{q}_i\right\|^2 + \left\|\boldsymbol{k}_j\right\|^2\right) / 2\sigma^2\right] \exp\left(\boldsymbol{q}_i \boldsymbol{k}_j^\top / \sigma^2\right)}{\sum_j^N \exp\left[-\left(\left\|\boldsymbol{q}_i\right\|^2 + \left\|\boldsymbol{k}_{j'}\right\|^2\right) / 2\sigma^2\right] \exp\left(\boldsymbol{q}_i \boldsymbol{k}_j^\top / \sigma^2\right)}.\end{aligned}$$

If we further assume that the keys $\boldsymbol{k}_j$ are normalized (usually done to stabilize the training), the value of $\widehat{f}_\sigma\left(\boldsymbol{q}_i\right)$ then becomes

$$\widehat{f}_\sigma\left(\boldsymbol{q}_i\right) = \frac{\sum_j^N \mathbf{v}_j \exp\left(\boldsymbol{q}_i \boldsymbol{k}_j^\top / \sigma^2\right)}{\sum_{j'}^N \exp\left(\boldsymbol{q}_i \boldsymbol{k}_{j'}^\top / \sigma^2\right)} = \sum_{j=1}^N \operatorname{softmax}\left(\boldsymbol{q}_i^\top \boldsymbol{k}_j / \sigma^2\right) \mathbf{v}_j.$$

[10] https://arxiv.org/abs/2206.00206

References of the Transformers Section

- https://huggingface.co/course/
- Transformer with Fourier Integral Attentions, by Tan Nguyen et al., https://arxiv.org/pdf/2206.00206.pdf
- A Recipe for Training Neural Networks, by Andrej Karpathy, https://karpathy.github.io/2019/04/25/recipe/
- Speech and Language Processing (3rd ed. draft) by Dan Jurafsky and James H. Martin
  https://web.stanford.edu/~jurafsky/slp3/10.pdf
  https://web.stanford.edu/~jurafsky/slp3/11.pdf