Sequence-to-Sequence Models

Tianxiang (Adam) Gao

February 27, 2025

Outline

Recap: Recurrent Neural Networks

- **Audio Waveform**: A 1D array represents the amplitude of the sound over time, *e.g.*, 16kHz
- **One-Hot Encoding**: Each word in a **vocabulary** is a **binary** one-hot vector.
- **Challenges in Text Data**: Curse of dimensionality and long-run dependencies.
- **Language Models**: Assigns probabilities to a given sequence of words
- **Neural Language Model**: Model the probability distribution of the next word given the history:

$$\mathbb{P}(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_t) = f_{\boldsymbol{\theta}}(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_t).$$

  RNNs: Encode the history into a hidden state $\boldsymbol{h}_t$ updated by combing with the current word $\boldsymbol{x}_t$:

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_h h_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b}_h), \quad \text{and} \quad \hat{\boldsymbol{y}}_t = \text{softmax}(\boldsymbol{W}_y \boldsymbol{h}_t + \boldsymbol{b}_y)$$

  Training RNNs: backpropagation through time
  - Forward (simplified): $\boldsymbol{h}_t = \phi(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t)$
  - Backward (simplified): $d\boldsymbol{h}_t = \boldsymbol{W}_h^\top \left(\boldsymbol{\phi}'_{t+1} \odot d\boldsymbol{h}_{t+1}\right) + \boldsymbol{W}_y^\top \left(\boldsymbol{\sigma}'_t \odot d\boldsymbol{y}_t\right)$
- **Generation**: Sample the next word from the predicted probability distribution produced by RNNs.
- **RNN Types**: One-to-many, many-to-one, or many-to-many structures for different tasks.
- **Vanishing/Exploding Gradients**: $\boldsymbol{h}_t = \mathcal{O}\left(a^t\right)$ and $d\boldsymbol{h}_t = \mathcal{O}\left(b^{T-t}\right)$.

Recap: Recurrent Neural Networks

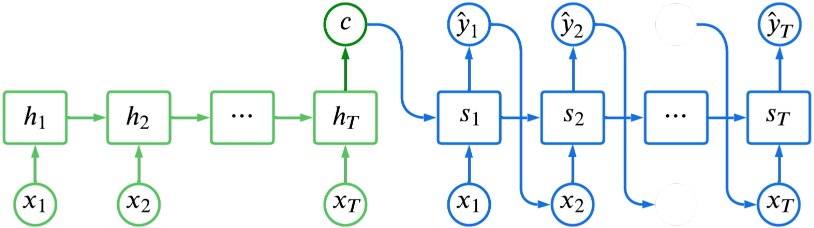
- **Gated Recurrent Unit (GRU)**: Gates helps maintain long-term dependencies:
$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_h(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{W}_x\boldsymbol{x}_t), \quad \text{and} \quad \boldsymbol{h}_t = \boldsymbol{z}_t\boldsymbol{h}_{t-1} + (1-\boldsymbol{z}_t) \odot \tilde{\boldsymbol{h}}_t$$
Long Short-Term Memory (LSTM): Use a cell state $\boldsymbol{c}_t$ to maintain long-term dependencies.
$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t, \quad \text{and} \quad \boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$
Bidirecitonal RNNs: The concatenated hidden state: $\boldsymbol{h}_t = [\overrightarrow{\boldsymbol{h}}_t, \overleftarrow{\boldsymbol{h}}_t]$
- **Deeper RNNs**: Each layer ℓ computes its hidden state using the hidden state from the layer $\ell - 1$:
$$\boldsymbol{h}_t^{\ell} = \tanh(\boldsymbol{W}_h^{(\ell)}\boldsymbol{h}_{t-1}^{(\ell} + \boldsymbol{W}_x^{(\ell)}\boldsymbol{h}_t^{(\ell-1)})$$
Drawbacks of One-Hot Representation: orthogonality and high dimensionality
- **Word Embedding**: Words are represented as **dense vectors** in a **lower-dimensional space**.
$$\boldsymbol{e} = \boldsymbol{E}\boldsymbol{x}$$
Continuous Bag of Words (CBOW): Predicts the target word given the context
- **Skip-Gram**: Predicts the context words given a target word.
- **Negative Sampling**: Reformulates the context-target predictions as a **binary** classification:
$$\mathcal{L}(\boldsymbol{E}) = -\sum_{(\boldsymbol{e}_c, \boldsymbol{e}_t)} \log \sigma(\boldsymbol{e}_t^\top \boldsymbol{e}_c) - \sum_{(\boldsymbol{e}_c, \tilde{\boldsymbol{e}}_t)} \log \sigma(-\boldsymbol{e}_t^\top \tilde{\boldsymbol{e}}_c),$$
where $\tilde{e}_t$ is **negative** target samples outside the context window.

Outline

1. Sequence-to-Sequence Models
2. Attention Mechanism
3. Transformer
4. Large Language Models: BERT

Seq2Seq: Sequence-to-Sequence Models

- **Define**: Sequence-to-Sequence (Seq2Seq) models are designed to handle tasks where both input and output are sequences of **variable** length, *e.g.*, machine translation or summarization.
- **Example**: "Jane visite l'Afrique en septembre." $\Longrightarrow$ "Jane visits Africa in September."
- **Encoder-Decoder Architecture**
  - **Encoder**: Processes and compresses the input sequence into a fixed-length **context vector**.
  - **Decoder**: Uses the context vector to generate the output sequence sequentially



Here c is the context vector summarizing the *entire* input sequence.

Cho, et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." EMNLP 2014.

Conditional Language Model

- **Language Model**: Assigns probabilities to a sequence of words $\{\boldsymbol{x}_t\} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$:

$$\mathbb{P}(\{\boldsymbol{x}_t\}) = \mathbb{P}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) = \mathbb{P}(\boldsymbol{x}_1) \cdot \prod_{t=2}^{T-1} \mathbb{P}(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t)$$

where each conditional probability is modeled as:

$$\mathbb{P}(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t) = f_{\boldsymbol{\theta}}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t)$$

Conditional Language Model: Assigns probabilities to a target sequence $\{\boldsymbol{y}_t\}$ *given* an input sequence $\{\boldsymbol{x}_t\}$:

$$\mathbb{P}(\{\boldsymbol{y}_t\} \mid \{\boldsymbol{x}_t\}) = \prod_{t=1}^{T'-1} \mathbb{P}(\boldsymbol{y}_{t+1} \mid \boldsymbol{y}_1, \ldots, \boldsymbol{y}_t, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$$

where the conditional probability for each word in the target sequence is:

$$\mathbb{P}(\boldsymbol{y}_{t+1} \mid \boldsymbol{y}_1, \ldots, \boldsymbol{y}_t, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) = g_{\boldsymbol{\phi}}(\boldsymbol{y}_t, \boldsymbol{s}_{t-1}, \boldsymbol{c})$$

with $(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T) \mapsto \boldsymbol{c}$ and $(\boldsymbol{y}_1, \cdots, \boldsymbol{y}_{t-1}) \mapsto \boldsymbol{s}_{t-1}$.

Cho, et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.” EMNLP 2014.

Beam Search

Greedy Search: Selects the most probable word at each step, which may lead to suboptimal sequences.
Beam Search: Tracks **multiple** high-probability sequences simultaneously to improve overall accuracy.

- **Initialization**: Start with the *seed* token and choose the top k words based on the probability distribution.
- **Expansion**: Predict the next word for each candidate, generating new sequences.
- **Pruning**: Keep the top k sequences with the highest log-probability scores, discarding the rest.

$$\begin{aligned}\log \mathbb{P}(\boldsymbol{y}_2, \boldsymbol{y}_1 \mid \boldsymbol{c}) &= \log \Big[\mathbb{P}(\boldsymbol{y}_2 \mid \boldsymbol{c}, \boldsymbol{y}_1) \cdot \mathbb{P}(\boldsymbol{y}_1 \mid \boldsymbol{c})\Big] \\ &= \log \mathbb{P}(\boldsymbol{y}_2 \mid \boldsymbol{c}, \boldsymbol{y}_1) + \log \mathbb{P}(\boldsymbol{y}_1 \mid \boldsymbol{c}).\end{aligned}$$

Repeat: Continue expanding and pruning until terminate

Remark

- **Beam Width (k)**: Requires k identical decoders to update candidate sequences simultaneously.
- **Advantages**: Balances between accuracy and computation; larger k increases accuracy but demands more computational resources.

Sutskever et al., "Sequence to Sequence Learning with Neural Networks," NeurIPS 2014.

Numerical Stability

Log-Probabilities for Stability:

- Since $\mathbb{P}(\cdot) \in [0, 1]$, the product of probabilities can approach zero, causing numerical instability.
- To address this, log-probabilities are used:

$$\begin{aligned}\boldsymbol{y}^* &= \underset{\boldsymbol{y}}{\operatorname{argmax}}\, \mathbb{P}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{T'} \mid \boldsymbol{c}) \\ &= \underset{\boldsymbol{y}}{\operatorname{argmax}}\, \frac{1}{T'} \sum_{t=1}^{T'} \log \mathbb{P}(\boldsymbol{y}_t \mid \boldsymbol{c}, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1})\end{aligned}$$

This transformation helps prevent underflow and enables stable computation of probabilities.

Error Analysis in Beam Search

Let $\boldsymbol{y}^*$ be the optimal sequence and $\hat{\boldsymbol{y}}$ the model's predicted sequence.

$$\boldsymbol{y}^* = \underset{\boldsymbol{y}}{\operatorname{argmax}}\ \mathbb{P}(\boldsymbol{y} \mid \boldsymbol{c}) = \mathbb{P}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{T'} \mid \boldsymbol{c})$$

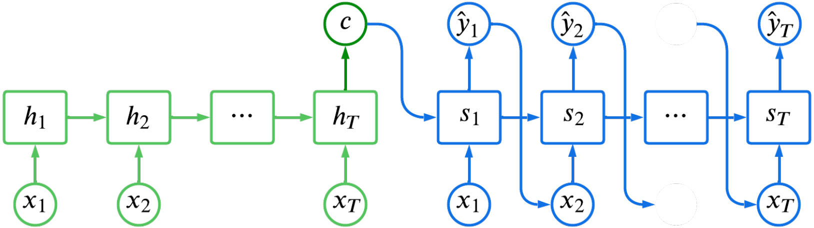
If $\mathbb{P}(\boldsymbol{y}^* \mid \boldsymbol{c}) > \mathbb{P}(\hat{\boldsymbol{y}} \mid \boldsymbol{c})$, then

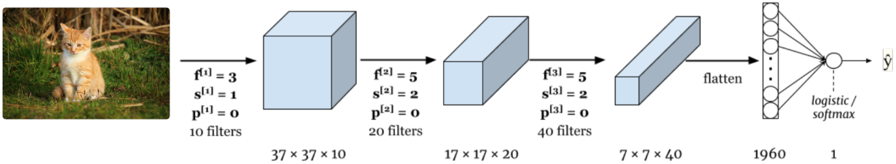
- Beam search pick $\hat{\boldsymbol{y}}$ but $\boldsymbol{y}^*$ is better, achieving higher $p(\boldsymbol{y} \mid \boldsymbol{c})$
- Increasing beam width can improve accuracy by exploring more potential sequences

If $\mathbb{P}(\boldsymbol{y}^* \mid \boldsymbol{c}) \leq \mathbb{P}(\hat{\boldsymbol{y}} \mid \boldsymbol{c})$, then

- $\boldsymbol{y}^*$ is a better translation than $\hat{\boldsymbol{y}}$, but RNN predicts $\mathbb{P}(\hat{\boldsymbol{y}} \mid \boldsymbol{c}) \geq \mathbb{P}(\boldsymbol{y}^* \mid \boldsymbol{c})$
- RNN module is at fault
- Increasing beam width won't help, as errors stem from model limitations.

Image Captioning: Using CNN as the Encoder





"A small orange kitten sits attentively on green grass, surrounded by natural, dried foliage in the background, giving a calm and serene outdoor setting."

Vinyals, et al. "Show and tell: A neural image caption generator." CVPR 2015.

BLEU Score: Bilingual Evaluation Understudy

BLEU Score: Bilingual Evaluation Understudy (BLEU) evaluates a machine-generated **candidate translation** $\hat{y}$ by comparing it to a *list* of reference translations $\{\boldsymbol{y}_1, \cdots, \boldsymbol{y}_M\}$.

- **Candidate Translation**: "the cat the cat on the mat"
- **Reference Translation 1**: "the cat is on the mat"
- **Reference Translation 2**: "there is a cat on the mat"

Precision: Measures how many n-grams in the candidate match the reference translations.

- **Candidate Bigrams**: {"the cat", "cat the", "the cat", "cat on", "on the", "the mat"}
- **Reference 1 Bigrams**: {"the cat", "cat is", "is on", "on the", "the mat"}
- **Reference 2 Bigrams**: {"there is", "is a", "a cat", "cat on", "on the", "the mat"}

$$\text{Precision} = \frac{\text{Number of matching n-grams}}{\text{Total n-grams in candidate}} = \frac{5}{6} \implies \textbf{Inflate the score!}$$

Papineni, et al. "BLEU: a method for automatic evaluation of machine translation." ACL 2002.

Modified Precision

Modified Precision: Limits the count of an n-gram to the maximum it appears in any reference.

- **Candidate Bigrams**: {"the cat", "cat the", "the cat", "cat on", "on the", "the mat"}
- **Reference 1 Bigrams**: {"the cat", "cat is", "is on", "on the", "the mat"}
- **Reference 2 Bigrams**: {"there is", "is a", "a cat", "cat on", "on the", "the mat"}

Bigrams	**Count**	**Matching**	**Clipped**
`the cat`	2	2	1
`cat the`	1	0	0
`cat on`	1	2	1
`on the`	1	3	1
`the mat`	1	2	1

The corresponding modified precision is given by:

$$\text{Modified Precision} = \frac{\text{Clipped number of matching n-grams}}{\text{Total n-grams in candidate}} = \frac{4}{6}$$

BLEU Score Formula

- For an n-gram, the **modified precision** p_n is defined as:

$$p_n = \frac{\text{Clipped number of matching } n\text{-grams}}{\text{Total } n\text{-grams in candidate}}$$

- The **BLEU score** is computed using the average of modified precisions, combined with a **brevity penalty (BP)** to penalize overly short translations:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\frac{1}{N}\sum_{n=1}^{N} \log p_n\right)$$

  where N is the maximum n-gram length considered (typically $N = 4$).

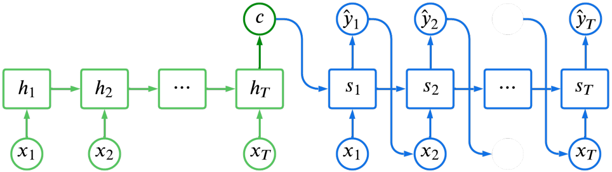
- The **Brevity Penalty (BP)** prevents high scores for short translations:

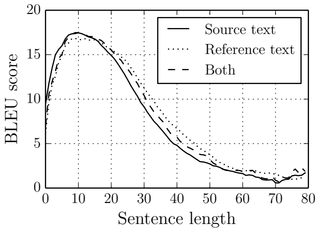
$$\text{BP} = \begin{cases} 1, & \text{if } c \geq r \\ \exp\left(1 - \frac{r}{c}\right), & \text{if } c < r \end{cases}$$

  where:
  - c is the length of the candidate translation.
  - r is the length of the reference translation.

Outline

Limitations of RNN Encoder-Decoder Framework





- **Encoder**: Process and compress the input sequence $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T\}$ into a context vector $\boldsymbol{c}$.

$$\boldsymbol{h}_t = f_{\boldsymbol{\theta}}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t), \qquad \boldsymbol{c} = \boldsymbol{h}_T.$$

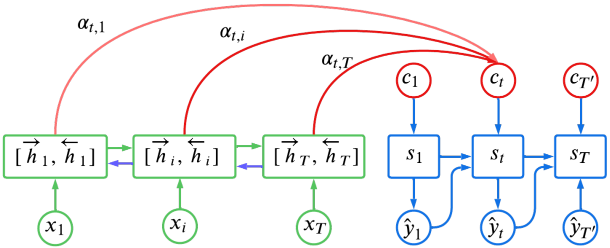
- **Decoder**: Uses the context vector $\boldsymbol{c}$ to generate the output sequence sequentially

$$\mathbb{P}(\boldsymbol{y}_t \mid \boldsymbol{x}, \boldsymbol{y}_1, \cdots, \boldsymbol{y}_{t-1}) = \mathbb{P}(\boldsymbol{y}_t \mid \boldsymbol{s}_t), \quad \text{where} \quad \boldsymbol{s}_t = g_{\boldsymbol{\phi}}(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{c})$$

Note: Encoding all information into a single vector $\boldsymbol{c}$ may cause information loss for longer sequences.

Bahdanau, et al. "Neural machine translation by jointly learning to align and translate." ICLR 2015

Distinct Context Vector in Attention Mechanism



- **Distinct Context Vector for Each Target Word**: Each target word $\boldsymbol{y}_t$ has a unique context vector $\boldsymbol{c}_t$, allowing the model to focus on relevant input parts.

$$\mathbb{P}(\boldsymbol{y}_t \mid \boldsymbol{x}, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1}) = \mathbb{P}(\boldsymbol{y}_t \mid \boldsymbol{s}_t), \quad \text{where} \quad \boldsymbol{s}_t = g_{\boldsymbol{\phi}}(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{c}_t)$$

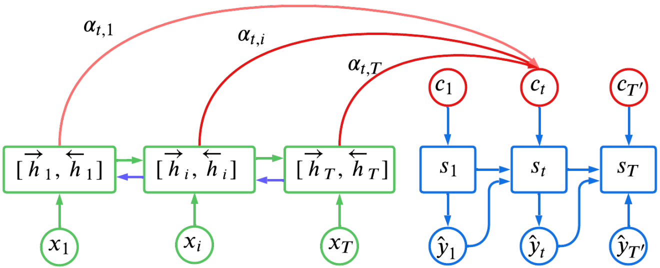
Context Vector Computation: The context vector $\boldsymbol{c}_t$ is computed as a weighted sum of encoder hidden states $\boldsymbol{h}_i$, tailored to the current decoding step.

$$\boldsymbol{c}_t = \sum_{i=1}^{T} \alpha_{t,i} \boldsymbol{h}_i$$

where **attention weights** $\alpha_{t,i}$ indicate the relevance of each hidden state $\boldsymbol{h}_i$ for generating $\boldsymbol{y}_t$.

Bahdanau et al., "Neural Machine Translation by Jointly Learning to Align and Translate," ICLR 2015

Attention Mechanism in Seq2Seq



- **Attention Weights**: Computed from an **alignment score** $e_{t,i}$ between the decoder's previous hidden state $\boldsymbol{s}_{t-1}$ and each encoder hidden state $\boldsymbol{h}_i$.

$$\alpha_{t,i} = \frac{\boldsymbol{e}_{t,i}}{\sum_j \boldsymbol{e}_{t,j}} \quad \text{where} \quad \boldsymbol{e}_{t,i} = a(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i) = \boldsymbol{v}^\top \tanh(\boldsymbol{W}\boldsymbol{s}_{t-1} + \boldsymbol{U}\boldsymbol{h}_i)$$

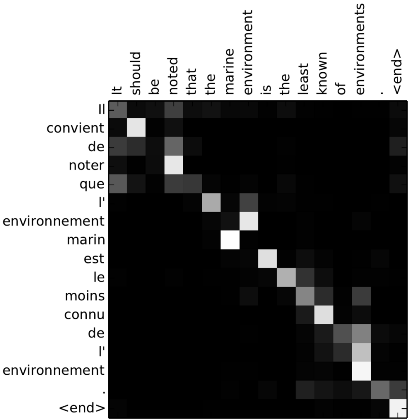
  is an **alignment model** implemented as an MLP, which is trained jointly with Seq2Seq.
- **Bidirectional Encoder**: The encoder uses a bidirectional RNN to capture both past and future context, enhancing comprehension of each input word's meaning.

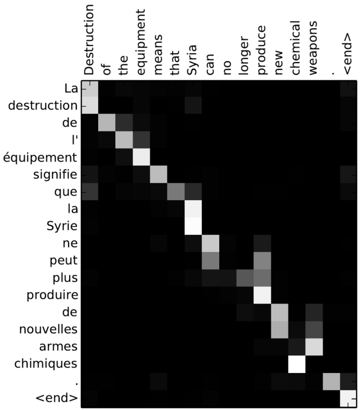
Bahdanau et al., "Neural Machine Translation by Jointly Learning to Align and Translate," ICLR 2015

Sample Alignments

- **Attention Weights**: Computed from an **alignment score** $e_{t,i}$ between the decoder's previous hidden state $\boldsymbol{s}_{t-1}$ and each encoder hidden state $\boldsymbol{h}_i$.

$$\alpha_{t,i} = \frac{\boldsymbol{e}_{t,i}}{\sum_j \boldsymbol{e}_{t,j}} \quad \text{where} \quad \boldsymbol{e}_{t,i} = a(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i)$$





Summary: Seq2Seq Models

- **Seq2Seq**: Use an RNN Encoder-Decoder architecture to handle tasks where both input and output are sequences.
- **Conditional Language Model**: The output sequence is generated sequentially based on the context vector that summarizes the input sequence
- **Beam Search**: Keeps multiple high-probability sequences to improve output quality.
- **BLEU Score**: A metric using modified precision to assess the accuracy of generated sequences.
- **Distinct Convex Vector**: A distinct context word $\boldsymbol{e}_t$ is used to generate each target word $\hat{\boldsymbol{y}}_t$

  $$\mathbb{P}(\boldsymbol{y}_t \mid \boldsymbol{x}, \boldsymbol{y}_1, \cdots, \boldsymbol{y}_{t-1}) = \mathbb{P}_\phi(\boldsymbol{y}_t \mid \boldsymbol{s}_t), \quad \text{where} \quad \boldsymbol{s}_t = g_\phi(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{c}_t)$$

  Attention Weights: The distinct context word $\boldsymbol{e}_t$ is a weighted sum of encoder hidden states:

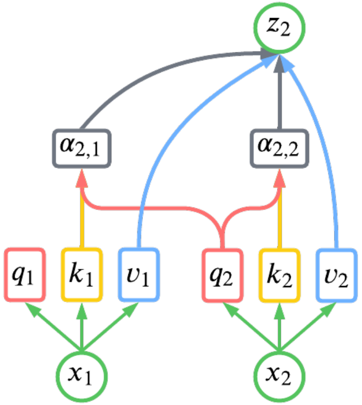
  $$\boldsymbol{c}_t = \sum_i \alpha_{t,i} \boldsymbol{h}_i,$$

  where $\alpha_{t,i} = \text{softmax}(\boldsymbol{e}_{t,i})$ are **attention weights**
- **Alignment Scores**: Alignment scores $\boldsymbol{e}_{t,i} = a(\boldsymbol{s}_{\cdot t-1}, \boldsymbol{h}_i)$ indicate relevance between encoder hidden states $\boldsymbol{h}_i$ and decoder states $\boldsymbol{s}_{t-1}$.

Outline

Self-Attention

Define: Self-attention creates a contextually enriched representation of each token by learning its relevance to all other tokens in the sequence.



- For each token, three vectors are computed:

$$\boldsymbol{q}_t = \boldsymbol{W}^q \boldsymbol{x}_t, \quad \boldsymbol{k}_t = \boldsymbol{W}^k \boldsymbol{x}_t, \quad \boldsymbol{v}_t = \boldsymbol{W}^v \boldsymbol{x}_t$$

where the query $\boldsymbol{q}_t$ interacts with keys $\boldsymbol{k}_i$ to measure relevance:

$$\boldsymbol{\alpha}_{t,i} \propto \boldsymbol{q}_t^\top \boldsymbol{k}_i, \quad \Rightarrow \quad \boldsymbol{\alpha}_t = \text{softmax}\left(\frac{\boldsymbol{K}\boldsymbol{q}_t}{\sqrt{d_k}}\right),$$

where

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{k}_1 & \cdots & \boldsymbol{k}_T \end{bmatrix}^\top$$

The new representation $\boldsymbol{z}_t$ is a weighted sum of value vectors $\boldsymbol{v}_i$:

$$\boldsymbol{z}_t = \sum_{i=1}^{T} \boldsymbol{\alpha}_{t,i} \boldsymbol{v}_i$$

Vaswani et al., "Attention is All You Need," NeurIPS 2017

Self-Attention: Matrix Form

- Define matrices:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^{q\top}, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{k\top}, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^{v\top}$$

  where

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{q}_1 & \cdots & \boldsymbol{q}_T \end{bmatrix}^\top, \quad \boldsymbol{K} = \begin{bmatrix} \boldsymbol{k}_1 & \cdots & \boldsymbol{k}_T \end{bmatrix}^\top, \quad \boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1 & \cdots & \boldsymbol{v}_T \end{bmatrix}^\top$$

  The attention weights are computed as:

$$\begin{bmatrix} \boldsymbol{\alpha}_1 & \cdots & \boldsymbol{\alpha}_T \end{bmatrix} = \text{softmax}\left(\frac{\boldsymbol{K}\boldsymbol{Q}^\top}{\sqrt{d_k}}\right)$$

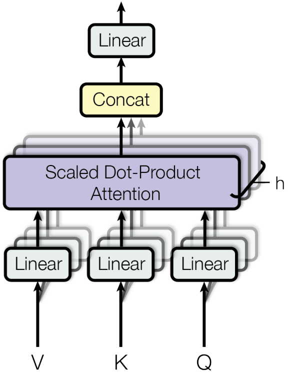
  The new representation $\boldsymbol{Z}$ is then:

$$\boldsymbol{Z} = \text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}$$

Vaswani et al., "Attention is All You Need," NeurIPS 2017

Multi-Head Attention

Definition: Multi-head attention extends self-attention by allowing **multiple heads** to focus on **different aspects** of each token, capturing diverse patterns and dependencies across the sequence.



- Each head produces an independent attention output $\boldsymbol{Z}_h$:

$$\boldsymbol{Z}_h = \text{Attention}(\boldsymbol{Q}_h, \boldsymbol{K}_h, \boldsymbol{V}_h),$$

  where $\boldsymbol{Q}_h = \boldsymbol{Q}\boldsymbol{W}_h^q$, $\boldsymbol{K}_h = \boldsymbol{K}\boldsymbol{W}_h^k$, and $\boldsymbol{V}_h = \boldsymbol{V}\boldsymbol{W}_h^v$.
- Head outputs are concatenated and linearly transformed to form the final representation:

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{Z}_1 & \cdots & \boldsymbol{Z}_H \end{bmatrix} \boldsymbol{W}_o^\top$$

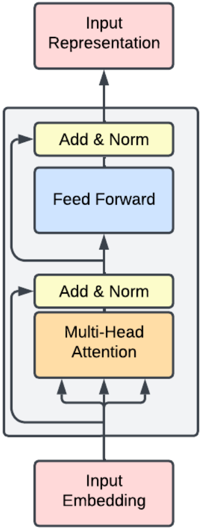
  where $\boldsymbol{W}_o$ is the output projection and H is the number of heads.
- **In Matrix Form**: With $\boldsymbol{Q}_h, \boldsymbol{K}_h, \boldsymbol{V}_h$ for each head,

$$\boldsymbol{Z} = \text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) \in \mathbb{R}^{T \times d_{\text{model}}}$$

Note: The multi-head can be computed in **parallel**, each with complexity $\mathcal{O}(T^2 d)$.

Vaswani et al., "Attention is All You Need," NeurIPS 2017

Multi-Head Attention Layer: LayerNorm and FNN



Layer Normalization computes statistics across different **hidden units**:

- In an RNN or MLP, a hidden state update is given by:

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}, \quad \boldsymbol{h} = \tanh(\boldsymbol{z})$$

  where the pre-activation vector $\boldsymbol{z} \in \mathbb{R}^m$.

- The statistics are computed across the **hidden units**:

$$\boldsymbol{z}_i = \boldsymbol{w}_i^\top \boldsymbol{x}, \quad \mu = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{z}_i, \quad \sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{z}_i - \mu)^2}$$

  where $\boldsymbol{w}_i$ is the ith row of $\boldsymbol{W}$.

- Re-scale and shift the normalized pre-activation:

$$\boldsymbol{z}_{\text{norm}} = \frac{\boldsymbol{z} - \mu}{\sigma}, \quad \tilde{\boldsymbol{z}} = \boldsymbol{\alpha} \odot \boldsymbol{z}_{\text{norm}} + \boldsymbol{\beta}$$

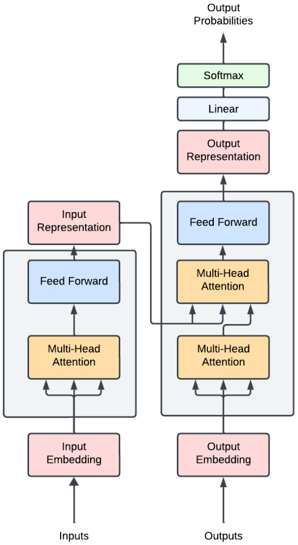
  where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are **trainable**.

Feed-Forward Layer captures non-linear relationships between tokens:

$$\text{FFN}(\boldsymbol{x}_t) = \text{ReLU}(\boldsymbol{x}_t \boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$$

Ba et al. "Layer normalization." NeurIPS 2016 in Deep Learning Symposium

Multi-Head Attention Encoder and Decoder Stacks



Encoder:

- **Input Embedding**: Converts input to dense word embeddings.
- **Multi-Head Attention**: Enhances token representations by attending to various parts of the sequence.
- **FFN**: Applies non-linear transformations to capture complex relationships between words.

Decoder:

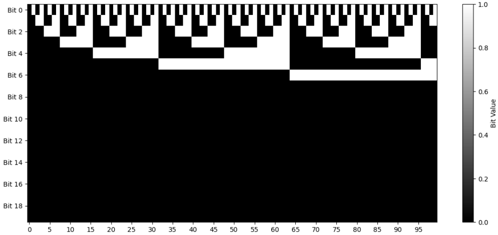
- **Output Embedding**: Converts the previously generated output (one-hot encoded) into dense word embeddings.
- **First Multi-Head Attention**: Refines the output embeddings or hidden states using self-attention.
- **Second Multi-Head Attention**: Uses the output of the first attention as the query Q, with K and V from the encoder's output, allowing the decoder to attend to the input sequence.
- **Final Output**: Computes the conditional probability of the next token through a linear layer and softmax.

Vaswani et al., "Attention is All You Need," NeurIPS 2017

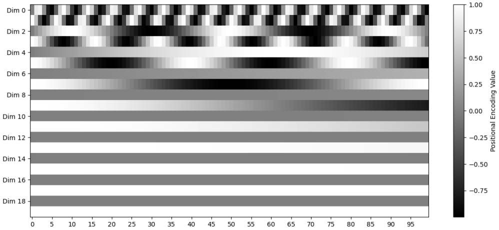
Positional Encoding

- Unlike RNNs, transformers do **not** inherently process tokens in sequence order.
- Positional Encoding is defined as:

$$\text{PE}_{\text{pos},2i} = \sin\left(\text{pos}/10000^{\frac{2i}{d_{\text{model}}}}\right), \quad \text{and} \quad \text{PE}_{\text{pos},2i+1} = \cos\left(\text{pos}/10000^{\frac{2i}{d_{\text{model}}}}\right)$$



Binary Representation



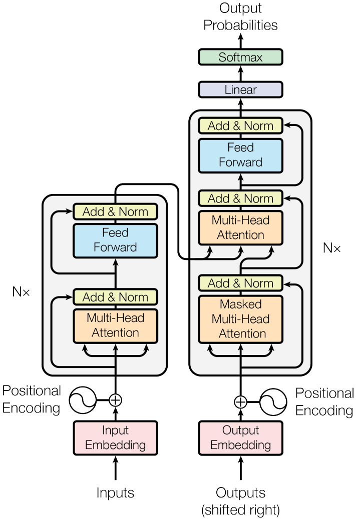
Position Encoding

Key Properties

- Provides unique and consistent representation for each position.
- Represents positions in a low-dimensional subspace.
- Enables linear transformations for relative positioning, *i.e.*, $\exists$ a linear M_k s.t. $M_k \text{PE}_{\text{pos}+k} = \text{PE}_{\text{pos}}$.

Amirhossein Kazemnejad's Blog: Positional Encoding in Transformers

Transformer



Training Process: Teacher Forcing

- During training, the **ground truth (actual) previous token** is fed into the decoder.
- Masking ensures only past and current tokens are visible, preserving autoregressive properties.
- Cross-entropy loss is used to compare the predicted probability distribution with the true token.

Loss Function: Cross-Entropy

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T} y_t \log \mathbb{P}(\hat{y}_t)$$

- y_t: True one-hot encoded token.
- $\mathbb{P}(\hat{y}_t)$: Predicted probability for the token at time t.

Vaswani et al., "Attention is All You Need," NeurIPS 2017

Summary

- **Self-attention** refines the representation of each token by learning its relevance to other tokens using **query-key** pairs.
- **Multi-head self-attention** captures **different aspects** of each token, enhancing the overall representation.
- **Layer normalization** computes statistics **across hidden units** to stabilize information propagation.
- **Positional encoding** adds **order** information to word embeddings, enabling the model to learn relative positioning through a linear transformation.
- **Encoder-decoder attention** refines the output representation by attending to the input sequence representation.
- The **Transformer** uses **teacher forcing** with cross-entropy loss to facilitate effective training.

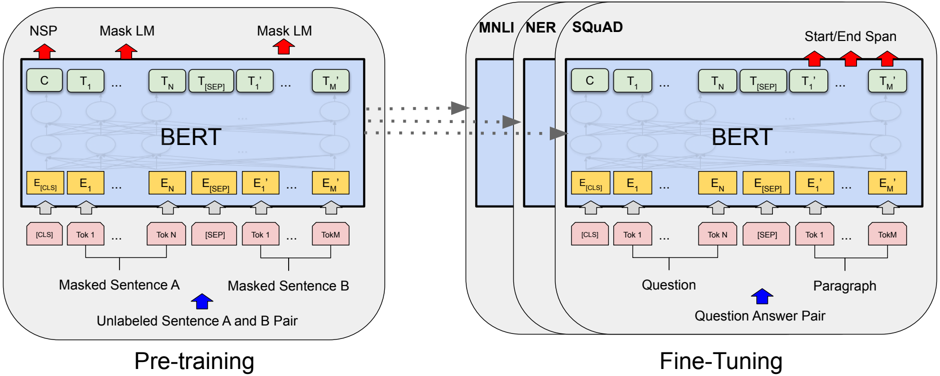
Outline

BERT

BERT: Encoder-Only

Definition: BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers.

- Utilizes only the **encoder** part of the Transformer architecture.
- Designed for pre-training on large corpora and fine-tuning on downstream NLP tasks.



Model Details: BERT_Large, with 340 million parameters, was trained on TPU v3 pods over 4 days using the BooksCorpus (800 million words) and English Wikipedia (2.5 billion words) datasets.

Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," NAACL 2019.

BERT: Pre-training

Masked Language Modeling (MLM): Randomly masks 15% of tokens in the input sequence and predicts masked tokens based on context.

- **Problem**: `[MASK]` token never used in finite-tuning
- **Solution**: Do not always replace selected words with `[MASK]`, e.g., `my dog is hairy`
  - 80% of the time: Replace the word with the `[MASK]` token, e.g., `my dog is [MASK]`
  - 10% of the time: Replace the word with a random word, e.g., `my dog is apple`
  - 10% of the time: Keep the word unchanged, e.g., `my dog is hairy`.

Next Sentence Prediction (NSP): Predicts if the second sentence follows the first, using the hidden state of the **[CLS]** token for binary classification (*IsNext* or *NotNext*).

- Input=`[CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]`, Label=`IsNext`
- Input=`[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]`, Label=`NotNext`

Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," NAACL 2019.

BERT: Performance on SQuAD 1.1

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**