Deep Learning

Class III: Sequence Modelling

Hédi Hadiji

Université Paris-Saclay - CentraleSupelec

hedi.hadiji@l2s.centralesupelec.fr

October, 2025

Table of Contents

Last time

Last time:

- Convolutional Neural Networks

Today: Sequential Data and Attention

Extra material

- This visualisation
- This video from 3B1B

Table of Contents

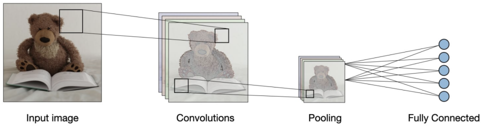
Deep Learning Paradigm

Given a dataset of input-output pairs:

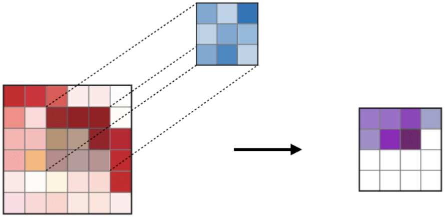
- Define a loss function.
- Define **functions** with **trainable parameters**.
- Optimize parameters with **Stochastic Gradient Descent** in order to minimize the loss
- Find architectures that capture useful structure of the data: convolutions will efficiently find local patterns in images/audio.
- Let SGD compute the actual patterns that are useful for the task.

Convolutional Neural Networks

Capture local patterns with **convolutions**.



Each convolutional layer applies multiple **kernels** (filters)



Visualization too

Table of Contents

Today: Sequential Data

Many forms of data are naturally represented as sequences:

- Text: words in a sentence, characters in a document.
- Audio: waveform samples, spectrogram frames.
- Time series: stock prices, sensor readings, weather data, ECG
- Video: frames in a video clip.
- Biological sequences: DNA, protein sequences.

Generically, we will call each unit element of the sequence a **token** (word, frame, sample, etc.).

Tasks

- Autoregressive prediction (next-word prediction): predict $p(x_t \mid x_{<t})$
- Sequence-to-sequence: generate an output sequence from an input sequence (e.g., translation, summarization, speech-to-text)
- Classification: sentiment analysis, activity recognition
- Sequence labeling: part-of-speech tagging, named entity recognition (e.g. split a conversation between speakers)
- Anomaly detection: identify unusual patterns in time series data, e.g., fraud detection

Limits of CNNs for Sequence Modelling

CNNs can be applied to sequential data and they are good for

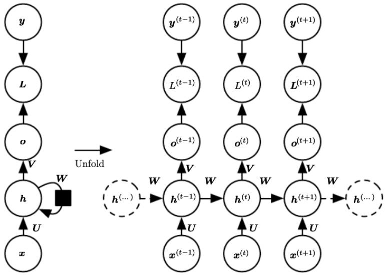
- **Parallelism:** Convolutions process all positions simultaneously, enabling faster training than RNNs.
- **Local pattern detection:** Excellent at extracting local motifs (e.g., phonemes, edges, n-grams) from raw sequences.
- **Compositional hierarchy:** Stacking layers builds progressively larger receptive fields and higher-level representations.

Limits of CNNs for Sequence Modelling

But have limitations:

- **Long-range dependencies:** Convolutions have a fixed receptive field; modeling long-term relations requires many layers or large kernels.
- **Non-stationary patterns:** Convs assume translation invariance; sequences whose statistics change over time (e.g., drifting topics, tempo changes) violate this assumption.
- **Order sensitivity:** CNNs are less precise with absolute positions or ordering unless positional encodings or padding tricks are added.
- **Variable-length context:** Fixed kernel sizes struggle when dependencies vary in length (some short-term, others long-term).
- **Inefficient for global aggregation:** Tasks needing global context (e.g., summarization, next-token prediction with far dependencies) are poorly handled without attention-like mechanisms.

Early Models for Sequence Data

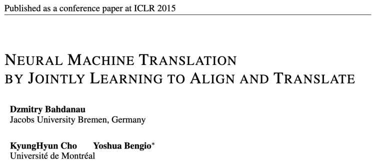


- Early models: Recurrent Neural Networks (RNNs) (e.g. LSTM, GRU).
- Process tokens one at a time, maintaining a hidden memory state.
- Update hidden state based on current input and previous state.
- Limitations:
  - Sequential computation (no parallelism)
  - Vanishing / exploding gradients
  - Difficulty with long-range dependencies

Table of Contents

The Attention Mechanism: History

Invented in 2015 for machine translation (Bahdanau et al., 2015).

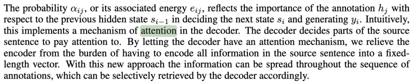

Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio**[*]
Université de Montréal

Mechanism used jointly with RNNs to allow the model to focus on relevant parts of the input sequence when generating each output token.


The probability α_{ij}, or its associated energy e_{ij}, reflects the importance of the annotation h_j with respect to the previous hidden state s_{i-1} in deciding the next state s_i and generating y_i. Intuitively, this implements a mechanism of attention in the decoder. The decoder decides parts of the source sentence to pay attention to. By letting the decoder have an attention mechanism, we relieve the encoder from the burden of having to encode all information in the source sentence into a fixed-length vector. With this new approach the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly.

Attention is All You Need

In 2017, Vaswani et al. remove recurrence and convolutions entirely.

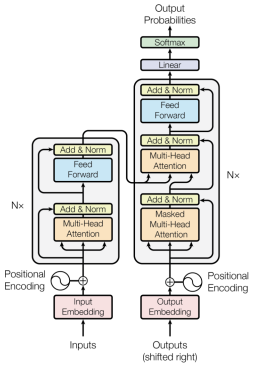


Figure 1: The Transformer - model architecture.

Transformers are now the dominant architecture for sequence modelling.

The Idea of Attention

In a given task (e.g., translation), not all input tokens are equally relevant to predict the current output token.

Instead of relying solely on the last hidden state of an RNN (which compresses all past information), attention allows the model to **dynamically focus** on different parts of the input sequence.

Canonical Setting: Next Character Prediction

Dataset is a corpus of texts, represented as sequences of characters.

Goal: predict the next character given all previous characters. e.g.

Input: `The cat sat on the ma` **Target:** `t`

Possible tokens (**vocabulary**): a-z, A-Z, punctuation, special tokens (space, start/end of sequence).

Prediction: a probability distribution p_t over possible next characters.

Loss: cross-entropy between predicted and true next character.

$$-\log p_t(\texttt{t})$$

First Step: Token Embeddings

Give each possible token a vector representation (embedding) in $\mathbb{R}^d$.

Embeddings are learned during training: we have an embedding matrix of size $|\text{vocab}| \times d$ of trainable parameters.

From now on, the sequence of tokens is represented as a sequence of embeddings:

$$X = [x_1, x_2, \dots, x_T], \quad x_i \in \mathbb{R}^d$$

In the next slides we define a function (layer) that takes as input a sequence of token embeddings and computes a new representation for each token that accounts for how much it attends to other tokens in the sequence.

(Self-)Attention Mechanism

For all token embedding $\in \mathbb{R}^d$, compute three vectors (Database analogy)

- **Keys:** unique identifiers for each record (token). $K \in \mathbb{R}^{d_k}$.
- **Queries:** requests to retrieve relevant records based on similarity to them. $Q \in \mathbb{R}^{d_k}$.
- **Values:** associated data for each record. $V \in \mathbb{R}^{d_v}$.

Idea : for each query, compute similarity to all keys, use these scores to weight the values and produce an output vector.

The Attention Formula

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

$QK^\top$: similarity scores between queries and keys.

Example Attention Scores with 3 Tokens

	K_1	K_2	K_3
Q_1	$Q_1 \cdot K_1$	$Q_1 \cdot K_2$	$Q_1 \cdot K_3$
Q_2	$Q_2 \cdot K_1$	$Q_2 \cdot K_2$	$Q_2 \cdot K_3$
Q_3	$Q_3 \cdot K_1$	$Q_3 \cdot K_2$	$Q_3 \cdot K_3$

- Each Q_i and K_j are vectors, so the entries are dot products.
- $Q_i \cdot K_j$ indicates how much token i contributes to the output when processing token j. i.e. **'How much token j attends to token i.'**

Example Attention Scores with 3 Tokens

Apply scaling by $\frac{1}{\sqrt{d_k}}$ to avoid large dot products and row-wise Softmax normalization to get attention weights.

	K_1	K_2	K_3
Q_1	$w_1^{(1)}$	$w_2^{(1)}$	$w_3^{(1)}$
Q_1	$w_1^{(2)}$	$w_2^{(2)}$	$w_3^{(2)}$
Q_1	$w_1^{(3)}$	$w_2^{(3)}$	$w_3^{(3)}$

- Normalize the rows with softmax to get attention weights: 1rst row measure how much token 1 attends to all others.
- Each row sums to 1.

Example Attention Mechanism with 3 Tokens

Compute the new embeddings as weighted means of the value vectors

$$\begin{pmatrix} w_1^{(1)} & w_2^{(1)} & w_3^{(1)} \\ w_1^{(2)} & w_2^{(2)} & w_3^{(2)} \\ w_1^{(3)} & w_2^{(3)} & w_3^{(3)} \end{pmatrix} \times \begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} = \begin{pmatrix} w_1^{(1)} V_1 + w_2^{(1)} V_2 + w_3^{(1)} V_3 \\ w_1^{(2)} V_1 + w_2^{(2)} V_2 + w_3^{(2)} V_3 \\ w_1^{(3)} V_1 + w_2^{(3)} V_2 + w_3^{(3)} V_3 \end{pmatrix}$$

- Each new token embedding is a weighted average of all value vectors.
- Weights determined by attention scores between queries and keys.

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

Deep Learning Philosophy:

Summary up to now: An attention layer maps a sequence of token embeddings to a new sequence of token embeddings which incorporate informations from the entire sequence. The structure of the computation favors interactions between tokens with similar keys and queries.

Define linear maps that compute keys, queries, and values from input embeddings:

$$\text{to\_key} \in \mathbb{R}^{d,d_k}, \quad \text{to\_queries} \in \mathbb{R}^{d,d_k}, \quad \text{to\_values} \in \mathbb{R}^{d,d_v},$$

Make them **trainable parameters** of the model.

Let the model learn to compute useful keys, queries, and values for the task at hand. (and embeddings too!)

Attention: Complexity

$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

- Computing $QK^\top$ requires $\mathcal{O}(T^2 d_k)$ operations.
- Multiplying by V requires $\mathcal{O}(T^2 d_v)$ operations.
- Total complexity: $\mathcal{O}\big(T^2(d_k + d_v)\big)$.

The computational cost of an attention layer grows quadratically with the sequence length T.

Attention summary

- Models long-range dependencies by letting each token attend to the whole sequence.
- Fully parallelizable across positions: no need to scan sequentially like RNNs.
- Produces context-aware token representations (outputs are weighted mixtures of value vectors).
- Computational and memory cost scale quadratically with sequence length ($\mathcal{O}(T^2)$).
- Impractical for very long sequences.

Table of Contents

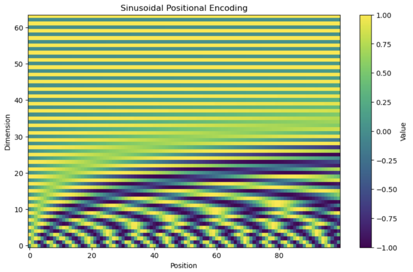
Positional Encodings

- Attention layers are **permutation invariant**: they do not know the order of tokens.
- To inject order, add a **positional encoding** to each input embedding:

$$z_t = x_t + p_t$$

- One common choice (Vaswani et al., 2017):

$$p_t^{(2i)} = \sin\left(\frac{t}{10000^{2i/d}}\right), \quad p_t^{(2i+1)} = \cos\left(\frac{t}{10000^{2i/d}}\right)$$



Multi-Head Attention

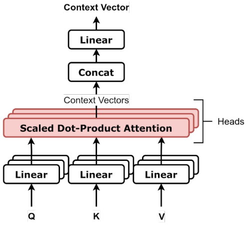
- Instead of a single attention mechanism, we use several in parallel.

$$\mathrm{MHA}(Q, K, V) = \mathrm{Concat}(H_1, \dots, H_h)\, W^O$$

- Each head *i* attends to different aspects of the input:

$$H_i = \mathrm{softmax}\left(\frac{QW_i^Q (KW_i^K)^\top}{\sqrt{d_k}} \right) VW_i^V$$

- Multiple **representation subspaces** and improves expressivity.



Normalization Layers

- Transformers rely on **Layer Normalization** for stable training.

$$\mathrm{LayerNorm}(x) = \frac{x - \mu(x)}{\sigma(x)} \odot \gamma + \beta$$

  where $\gamma, \beta \in \mathbb{R}^d$ are trainable parameters and

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

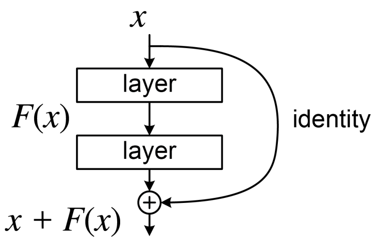
  (operations are done on the feature dimension)
- Stabilizes gradients and helps residual connections work effectively.

Residual Connections

- Deep networks may suffer from **vanishing gradients** and training instability.
- Idea: add a **shortcut connection** that skips one or more layers.

$$y = \mathcal{F}(x, W) + x$$

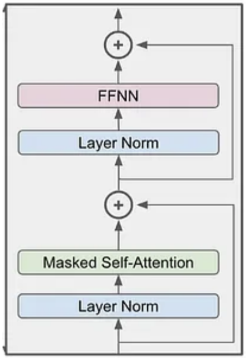
where $\mathcal{F}$ is the transformation (e.g., attention + feedforward).



Transformer Block

A Transformer block consists of:

- Multi-head self-attention layer
- Add & LayerNorm
- Feedforward Network (two linear layers with GELU activation)
- Add & LayerNorm



Next-token Prediction with Attention

- Transformers are sequence-to-sequence models. To use them for next-token prediction, we need to ensure **causality**.
- To prevent “peeking” at future tokens, we apply a **causal mask** to the attention scores:

$$\text{mask}(i,j) = \begin{cases} 0, & j \leq i \\ -\infty, & j > i \end{cases}$$

$$w_j^i \leftarrow w_j^i + \text{mask}(i,j)$$

- During training, the mask ensures that token x_t only attends to $x_1, \ldots, x_t$.

Example Attention Scores with 3 Tokens

	K_1	K_2	K_3
Q_1	$Q_1 \cdot K_1$	$-\infty$	$-\infty$
Q_2	$Q_2 \cdot K_1$	$Q_2 \cdot K_2$	$-\infty$
Q_3	$Q_3 \cdot K_1$	$Q_3 \cdot K_2$	$Q_3 \cdot K_3$

$$\xrightarrow[\text{Softmax}]{}$$

	K_1	K_2	K_3
Q_1	$w_1^{(1)}$	0	0
Q_2	$w_1^{(2)}$	$w_2^{(2)}$	0
Q_3	$w_1^{(3)}$	$w_2^{(3)}$	$w_3^{(3)}$

- Each token can only attend to previous tokens (including itself).
- Masking at the dot-product level saves some computation: no need to normalize twice.

Table of Contents

A Full Pipeline for Next-Token Prediction

Given a corpus of text that contains sequences of tokens:

$$(x_1^{(i)}, x_2^{(i)}, \dots, x_{T_i}^{(i)}), \quad i = 1, \dots, N$$

Given one input sequence of embeddings:

- Apply transformer architecture with L layers of attention blocks. Obtain a new sequence of embeddings with contextual information.
- Plug a **prediction head**, a linear layer that maps the final embeddings to vocabulary size.

Predict targets:

$$(x_2^{(i)}, x_3^{(i)}, \dots, x_{T_i}^{(i)}), \quad i = 1, \dots, N$$

Using cross-entropy loss between predicted and true next tokens.

Conclusion

TP: Implement a simple transformer for next-token prediction on text data.

Next time: More transformers: full architecture, images, embeddings, efficient attention...