Attention Networks and Diffusion Models
Lecture 2

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

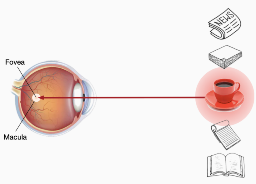
Bielefeld University

April 18, 2023

Attention Networks: Tutorial II

Attention: Reminder

Attention: Nonvolitional Cues

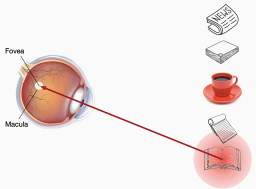


Nonvolitional cue: eye directs attention *non-voluntarily* to red coffee cup

From `https://d2l.ai`

- Nonvolitional cues based on saliency / conspicuity of objects
- *Example:*
  - Papers on desk black and white
  - Coffee cup red
  - *Consequence:* Eye "sees" coffee cup first
    ☞ Person grabs and drinks coffee

Attention: Volitional Cues



Deliberately searching for entertainment, eye *voluntarily* directs attention to book

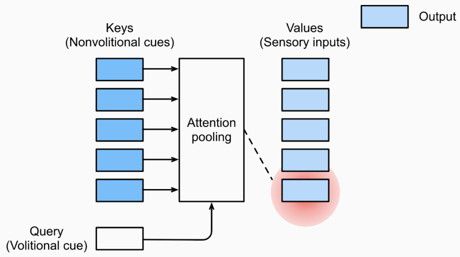
From `https://d2l.ai`

- Done with coffee, brain wants entertainment
- *Consequence:* Eye “sees” book in a deliberate attempt
- *Task-oriented search:*
  - Brain pre-trained to recognize objects that promise entertainment
  - Selection of book under full cognitive and volitional control

Queries, Keys and Values

ATTENTION: QUERIES, KEYS AND VALUES I

MOTIVATION

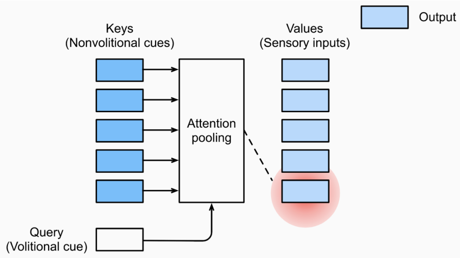


Attention pooling: integrating queries with keys (weights) and values (input)

- There are no queries in feed forward neural networks
- Feedforward neural networks reflect non-volitional attention
- *Goal:* Model volitional attention cues and integrate them appropriately
- Model patterned after database searches

Attention: Queries, Keys and Values II

Solution

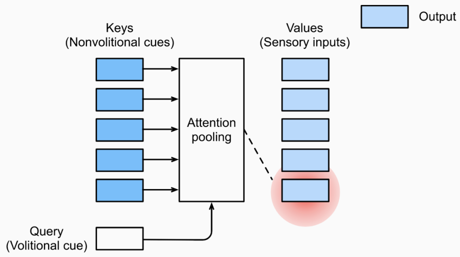


Attention pooling: integrating queries with keys (weights) and values (input)

- Ordinary neurons linearly combine weights w_i with input values x_i
- Weights w_i reflect non-volitional cues. The larger w_i
  - the more non-volitional attention directed to it
  - the higher x_i is rated for computing output

Attention: Queries, Keys and Values III

Attention Pooling

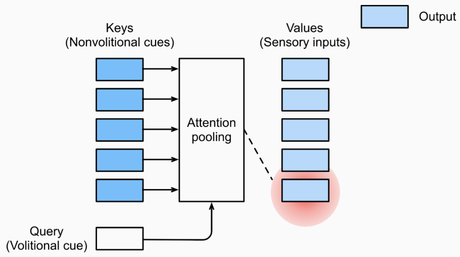


Attention pooling: integrating queries with keys (weights) and values (input)

- Volitional cue modeled by *query*
- Non-volitional cues correspond to the *keys* ($\leftrightarrow$ weights in ordinary NN)
- Matching *queries* with *keys* yields *attention weights*

ATTENTION: QUERIES, KEYS AND VALUES IV

ATTENTION POOLING



Attention pooling: integrating queries with keys (weights) and values (input)

- Attention weight reflects relevance of input relative to volitional cue
- *Attention pooling:* Compute "attention weighted" sum of values
- *Output* dominated by values whose keys match query well

Attention Pooling

ATTENTION POOLING: FORMAL SUMMARY

- Let $\mathbf{q} \in \mathbb{R}^q$ be a query and $(\mathbf{k}_1, \mathbf{v}_1), ..., (\mathbf{k}_m, \mathbf{v}_m), \mathbf{k}_i \in \mathbb{R}^k, \mathbf{v}_i \in \mathbb{R}^v$ be m key-value pairs
- The *attention pooling* f computes as

$$f(\mathbf{q}, (\mathbf{k}_1, \mathbf{v}_1), ..., (\mathbf{k}_m, \mathbf{v}_m)) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i \in \mathbb{R}^v \tag{1}$$

- The *attention weight* $\alpha(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}$ computes as

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \text{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \tag{2}$$

- The *attention scoring function* $a(\mathbf{q}, \mathbf{k})$ maps two vectors to a scalar

$$a : \mathbb{R}^q \times \mathbb{R}^k \longrightarrow \mathbb{R} \tag{3}$$

Additive Attention Scoring

- Let $\mathbf{q} \in \mathbb{R}^q$ be a query and $\mathbf{k} \in \mathbb{R}^k$ be a key
- Let $\mathbf{W}_q \in \mathbb{R}^{h \times q}, \mathbf{W}_k \in \mathbb{R}^{h \times k}, \mathbf{w}_v \in \mathbb{R}^h$ collect learnable parameters
- The *additive attention scoring function* computes as

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{w}_v^T \tanh(\mathbf{W}_q \mathbf{q} + \mathbf{W}_k \mathbf{k}) \in \mathbb{R} \tag{4}$$

- *Interpretation:* (4) reflects running $\mathbf{q}, \mathbf{k}$ through MLP
    - *Input:* Concatenation of $\mathbf{q}$ and $\mathbf{k}$
    - One *hidden layer* of width h
    - Parameters from input to hidden layer are $\mathbf{W}_q, \mathbf{W}_k$
    - The activation function is tanh
    - Parameters from hidden to output layer captured by $\mathbf{w}_v$

Scaled Dot-Product Attention Scoring I

- Let $\mathbf{q}, \mathbf{k} \in \mathbb{R}^d$ be *equal-sized* query and key
- The *scaled dot-product attention scoring function* computes as

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k} / \sqrt{d} \tag{5}$$

- *Note:* Dot product $\mathbf{q}^T \mathbf{k}$ has mean 0 and variance d
  ☞ Dividing by $\sqrt{d}$ implies standard deviation of 1

Scaled Dot-Product Attention Scoring II

Minibatches:

- Computing attention for n queries and m keys at once
  - *Reminder:* m keys come paired with m values
- For queries $\mathbf{Q} \in \mathbb{R}^{n \times d}$, keys $\mathbf{K} \in \mathbb{R}^{m \times d}$, values $\mathbf{V} \in \mathbb{R}^{m \times v}$ compute

$$\mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V} \in \mathbb{R}^{n \times v} \tag{6}$$

- Each row in (6) reflects weighted sum of values
  - n different queries yield n different weighted sums

Multi-Head Attention

Multi-Head Attention I

- *Motivation:* Capture different attention mechanisms for same queries, keys, values
- *Biology:* The same idea can trigger several, different things
- *Practical Example:* Attend to both short- and long-range dependencies in sequential data
- *Question:* How to vary attention mechanisms in informed way?

Multi-Head Attention II

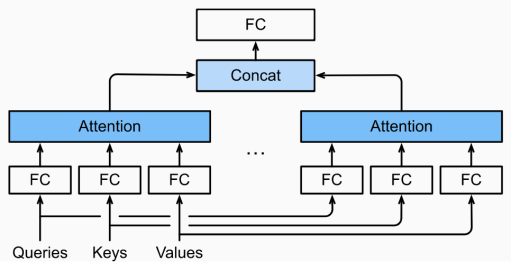
- *Question:* How to vary attention mechanisms in informed way?
- *Solution:*
  - Let h be intended number of attention mechanisms
  - Linearly transform queries, keys, values using h different sets of matrices $\mathbf{W}_i^{(q)}, \mathbf{W}_i^{(k)}, \mathbf{W}_i^{(v)}, i = 1, ..., h$
  - Run the h differently transformed queries, keys, values through attention pooling
  - Transformations $\mathbf{W}_i^{(q)}, \mathbf{W}_i^{(k)}, \mathbf{W}_i^{(v)}, i = 1, ..., h$ are learnt
  - The h attention pooling outputs are concatenated, and linearly transformed by another learned matrix $\mathbf{W}_o$
- Design is called *multi-head attention*
- Each of the h attention pooling outputs is referred to as a *head*

Multi-Head Attention III

- Let $\mathbf{q} \in \mathbb{R}^{d_q}, \mathbf{k} \in \mathbb{R}^{d_k}, \mathbf{v} \in \mathbb{R}^{d_v}$ be query, key, value
- Let $\mathbf{W}_i^{(q)} \in \mathbb{R}^{p_q \times d_q}, \mathbf{W}_i^{(k)} \in \mathbb{R}^{p_k \times d_k}, \mathbf{W}_i^{(v)} \in \mathbb{R}^{p_v \times d_v}$ collect learnable parameters
- f is attention pooling (1), using additive (4) or dot-product (5) scoring
- Each attention head is computed as

$$\mathbf{h}_i = f(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}) \in \mathbb{R}^{p_v} \tag{7}$$

Multi-Head Attention IV



From https://d2l.ai

- Attention heads:

$$\mathbf{h}_i = f(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}) \in \mathbb{R}^{p_v} \tag{8}$$

- Initial 'FC' layers reflect operations $\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}$
- 'Attention' layers reflect application of f to $\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}$

Multi-Head Attention V

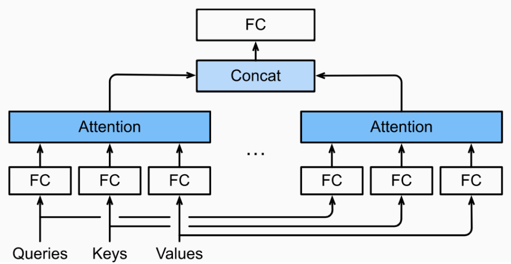
- Attention heads:

$$\mathbf{h}_i = f(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}) \in \mathbb{R}^{p_v} \tag{9}$$

- Let $\mathbf{W}_o \in \mathbb{R}^{p_o \times h p_v}$ collect further learnable parameters
- The multi-head attention output computes as

$$\mathbf{W}_o \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_h \end{bmatrix} \in \mathbb{R}^{p_o} \tag{10}$$

Multi-Head Attention VI



From https://d2l.ai

- Multi-head attention output computes as

$$\mathbf{W}_o[\mathbf{h}_1^T, ..., \mathbf{h}_h^T]^T \in \mathbb{R}^{p_o} \tag{11}$$

- 'Concat' layer reflects forming $[\mathbf{h}_1^T, ..., \mathbf{h}_h^T]$
- Final 'FC' layer reflects application of $\mathbf{W}_o$

Encoder-Decoder Architectures

Motivation: Sequence-2-Sequence Models

SEQUENCE-2-SEQUENCE MODELS I

- *Motivation:* Translate series of tokens into another series of tokens
- *Specific Example:* Translate sentences from one language to another
- *Challenges:*
  - Input and output differ in length
  - Sentences are unaligned (e.g. different grammar rules apply)
- Sequence-2-sequence models: neural networks accounting for this

Sequence-2-Sequence Models II

From `https://jalammar.github.io`

Sequence-2-Sequence Models III

From `https://jalammar.github.io`

Encoder-Decoder Architecture

Encoder-Decoder Architecture I

From `https://d2l.ai`

- *Encoder:*
  - Takes input sentence
  - Transforms it into *context state*
- *Decoder:*
  - Takes context state as input
  - Generates output sequence, token by token

ENCODER-DECODER ARCHTITECTURE II

From `https://jalammar.github.io`

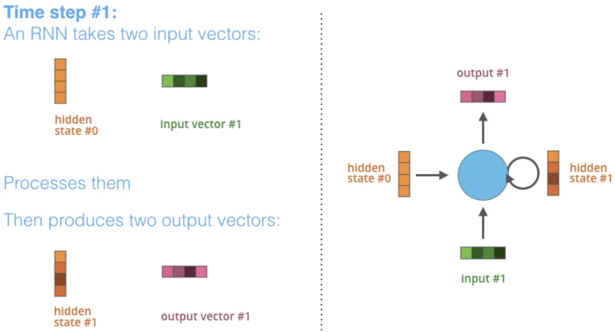
UNIVERSITÄT
BIELEFELD

Encoder-Decoder Architecture III

From `https://jalammar.github.io`

Encoder-Decoder using RNN's

ENCODER-DECODER: RNN REMINDER



From `https://jalammar.github.io`

Time step #1 shown; for time step $\#i$, $i \geq 1$ in general:

- RNN takes in hidden state $\#(i-1)$, input vector $\#i$
- RNN generates hidden state $\#i$, output vector $\#i$

Encoder-Decoder RNN I

From `https://jalammar.github.io`

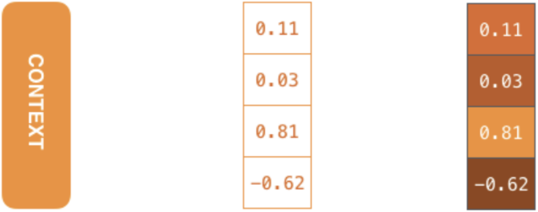
- *RNN Encoder:*
  - Uses sentence to translate as input
  - Generates new hidden state each time step; no output
- *RNN Decoder:*
  - Uses last hidden state of encoder as input
  - Output is translated sentence

Encoder-Decoder RNN II

From `https://jalammar.github.io`

- *Unrolled view:* Inputs and outputs per time step
- *Not shown:*
  - Encoder stops when receiving “end-of-sequence” $< eos >$ token
  - Decoder stops when generating “end-of-sequence”$< eos >$ token

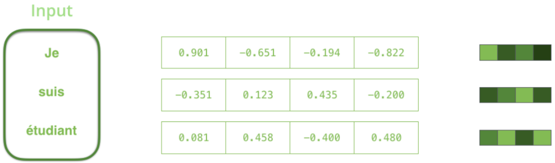
Encoder-Decoder: Context



From https://jalammar.github.io

- Context vector is a real-valued vector
- Dimension of context = # hidden units in encoder RNN

Encoder-Decoder: Word Embedding



From `https://jalammar.github.io`

- Tokens are embedded using *word embedding* techniques
- *Popular choice:* Word2Vec (e.g. 15.1 in `https://d2l.ai`)
- Typical sizes of embedding vectors: 200 to 300
- Excellent pre-trained embeddings available

RNN Encoder: Formal Description

- Let $x_1, ..., x_T$ be the input sequence, where x_t is t-th token
- Let $\mathbf{x}_t$ be feature vector of x_t, i.e. the embedding of x_t
- To generate hidden state t, the encoder computes at time step t:

$$f(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{12}$$

  where f expresses the transformation of the encoder's recurrent layer
- In general, the context variable $\mathbf{c}$ is computed as

$$\mathbf{c} = q(\mathbf{h}_1, ..., \mathbf{h}_T) \tag{13}$$

  where q is a customized function
- For example, often (e.g. in movies) $\mathbf{c} = q(\mathbf{h}_1, ..., \mathbf{h}_T) = \mathbf{h}_T$
- *Remark:* This refers to a unidirectional RNN
  ☞ bidirectional RNN's can be used as well

RNN Decoder: Formal Description

- Let $y_1, ..., y_{T'}$ be a *target output sequence*
  - *Training:* $y_1, ..., y_{T'}$ reflects true sequence
  - *Prediction:* $y_{t'+1}$ predicted based on $y_1, ..., y_{t'}$
- Let $\mathbf{c}$ be the context variable generated by encoder
- At time step $t' + 1$, decoder computes

$$\mathbf{P}(y_{t'+1} \mid y_1, ..., y_{t'}, \mathbf{c}) \tag{14}$$

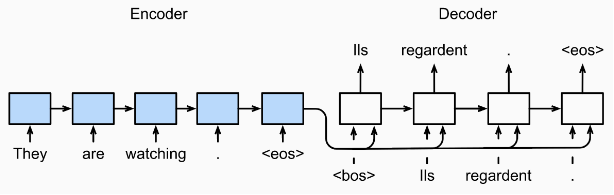
  for all possible $y_{t'+1}$

  - Given $y_{t'}$, the hidden state $\mathbf{s}_{t'-1}$ and $\mathbf{c}$, the RNN decoder computes

$$\mathbf{s}_{t'} = g(y_{t'}, \mathbf{c}, \mathbf{s}_{t'-1}) \tag{15}$$

  - Given $\mathbf{s}_{t'}$, one uses output layer and softmax operation to compute (14)

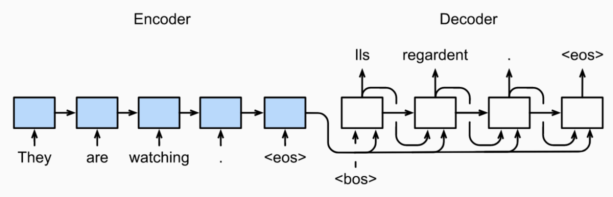
Training: Teacher Forcing



From `https://jalammar.github.io`

- Training uses correctly translated sequence as output target sequence
- *Teacher forcing:* Input and output shifted by one position relative to each other
  - $< \text{bos} >$ and $< \text{eos} >$ mean beginning and end of sentence, resp.
- Given all prior words, decoder RNN trained to translate next word

PREDICTION: TOKEN BY TOKEN



From https://d2l.ai

- Predicted token $y_{t'}$ from previous step fed into decoder as input
  - At the beginning, feed $<$ bos $>$ as input
- *Simple strategy:* Predict $y_{t'+1}$ that maximizes $\mathbf{P}(y_{t'+1} \mid y_1, \ldots, y_{t'}, \mathbf{c})$
  - *Beware:* Resulting $y_1, \ldots, y_{T'}$ may not maximize $\mathbf{P}(y_1, \ldots, y_{T'})$
  - More complex strategies available (e.g. `https://d2l.ai`, 10.8)
- When $<$ eos $>$ is predicted, output is complete

Attention II

Bahdanau Attention

Bahdanau Attention: Motivation

- Encoder-decoder architectures work well for short sentences
- Long, complex sentences:
  - Final encoder state too small to capture long sentence
  - *But*, final state complete and only source of information
- In 2014, Bahdanau suggested a model that
  - was inspired by the idea to align sequences / sentences
  - is differentiable
  - does not have the unidirectional alignment limitation
- When predicting a token, the model
  - only aligns (attends) to parts of input sequence deemed relevant
  - uses attention pattern to update current state before prediction
- Arguably, one of the most influential ideas in the last decade

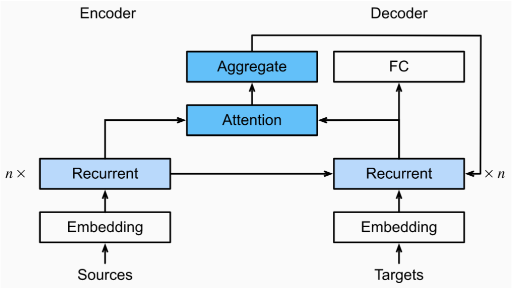
Bahdanau Attention: Formal Definition

- Let $\mathbf{h}_t$ be the hidden state of the encoder at time t
- Let $\mathbf{s}_{t'-1}$ be the hidden state of the decoder at time $t' - 1$
- Let $\mathbf{c}_{t'}$ be the context variable (i.e. state) after time t'
- Taking $\mathbf{s}_{t'-1}$ as query, and $\mathbf{h}_t$ as both key and value

$$\mathbf{c}_{t'} = \sum_{t=1}^{T} \alpha(\mathbf{s}_{t'-1}, \mathbf{h}_t)\mathbf{h}_t \tag{16}$$

  determines $\mathbf{c}_{t'}$ where T is the length of the input sequence
- α reflects the additive attention scoring function (4)
- One further proceeds using formulas (14),(15)

Bahdanau Attention: Schematic



Schematic of Bahdanau Attention

From `https://d2l.ai`

- One can integrate already generated tokens into attention (16): see `https://arxiv.org/pdf/1508.01211.pdf`

Self-Attention

Self-Attention: Definition

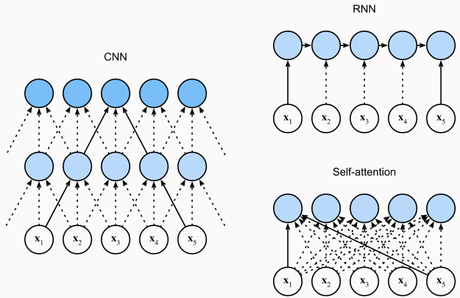
- Consider a sequence of tokens $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$
- Each token has its own query, key, and value
- Hence, each token can attend to each other token:
  - Pair the query vector with the key of the other token
  - This yields a weight for its own value
- Compute weighted sum of values as representation in next layer

Self-Attention: Formal Summary

- Consider a sequence of tokens $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$
- Replace $\mathbf{q}$ with $\mathbf{x}$ and both $\mathbf{k}_i, \mathbf{v}_i$ with $\mathbf{x}_i$ in (1)
- One obtains a new sequence $\mathbf{y}_1, ..., \mathbf{y}_n \in \mathbb{R}^d$ by

$$\mathbf{y}_i := f(\mathbf{x}_i, ((\mathbf{x}_1, \mathbf{x}_1), ..., (\mathbf{x}_n, \mathbf{x}_n))) = \sum_{j=1}^{n} \alpha(\mathbf{x}_i, \mathbf{x}_j)\mathbf{x}_j \in \mathbb{R}^d \tag{17}$$

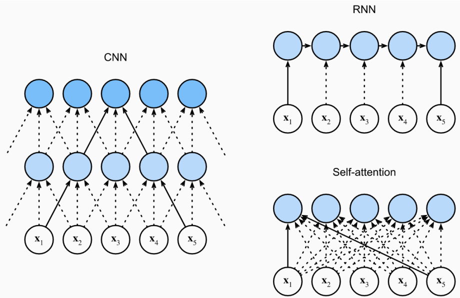
Computational Complexity: Comparison I



From `https://d2l.ai`

- Let n be the length of the sequence
- Let input/output tokens be represented by d-dimensional vectors
  - For CNN's, this agrees with the number of channels

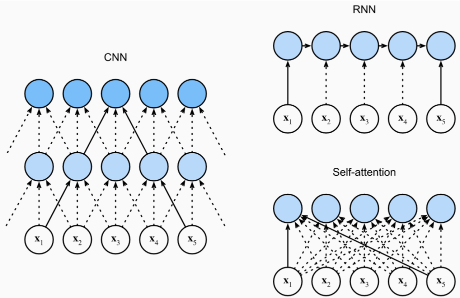
Computational Complexity: Comparison II



From `https://d2l.ai`

- *Computational complexity:* Number of arithmetic operations
- *Sequential operations:* Number of operations to be carried out consecutively
  - Sequential operations prevent parallelization

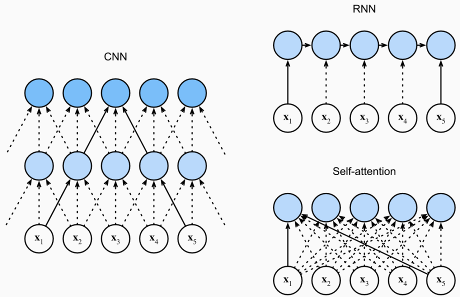
Computational Complexity: Comparison III



From `https://d2l.ai`

- *Maximum path length:* Maximum distance between two tokens
  - Distance measured in terms of edges in schematic
  - Long path length prevents mapping long-range dependencies

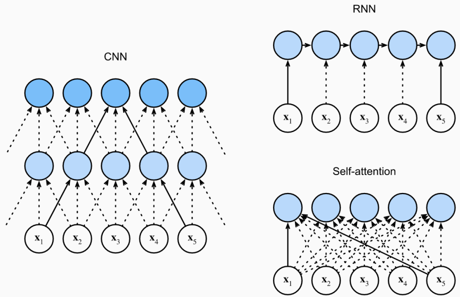
Computational Complexity: CNN's



From `https://d2l.ai`

- Let k be the filter size and d number of both input and output channels
- *Computational complexity:* $\mathcal{O}(knd^2)$
- *Sequential operations:* $\mathcal{O}(1)$
- *Maximum path length:* $\mathcal{O}(n/k)$

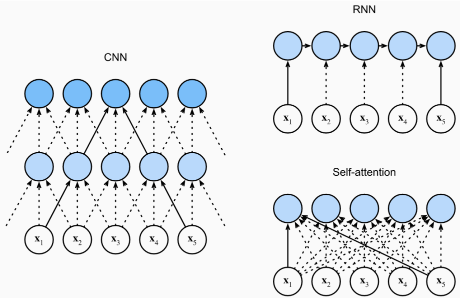
Computational Complexity: RNN's



From `https://d2l.ai`

- *Computational complexity:* $\mathcal{O}(nd^2)$; multiplying $d \times d$ weight matrix with d-dimensional hidden state
- *Sequential operations:* $\mathcal{O}(n)$
- *Maximum path length:* $\mathcal{O}(n)$

Computational Complexity: Self-Attention



From https://d2l.ai

- Queries, keys, values: $n \times d$-matrices
- *Computational complexity:* $\mathcal{O}(n^2 d)$
  - Scaled dot-product attention: multiply $n \times d$ with $d \times n$ with $n \times d$ matrix
  - Formula in compact form was $\text{softmax}(\frac{\mathbf{QK}^T}{\sqrt{d}})\mathbf{V}$
- *Sequential operations and maximum path length:* $\mathcal{O}(1)$

Thanks for your attention!