

Biological Applications of Deep Learning

Lecture 12

Alexander Schönhuth

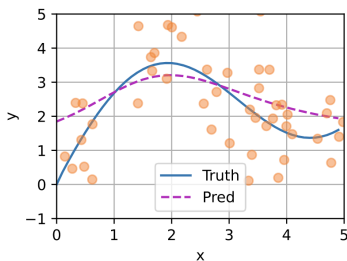


Bielefeld University
January 18, 2023

CONTENTS TODAY

- ▶ Attention Mechanisms I
 - ▶ Attention Scoring Functions
 - ▶ Multi-Head Attention
- ▶ Encoder-Decoder Architectures
 - ▶ *Motivation*: Sequence-2-Sequence Models
 - ↳ Translating Languages
 - ▶ Architectural Details
 - ▶ *Origin*: RNN based Architectures
 - ▶ Contexts and Embeddings
- ▶ Attention Mechanisms II
 - ▶ Bahdanau Attention
 - ▶ Self Attention

REMINDER: NADARAYA-WATSON KERNEL REGRESSION



From <https://d2l.ai>

- Using a Gaussian kernel yields (dashed pink curve)

$$\begin{aligned}\hat{f}(x) &= \sum_{i=1}^n \alpha(x, x_i) y_i = \sum_{i=1}^n \frac{\exp(-\frac{1}{2}(x - x_i)^2)}{\sum_{j=1}^n \exp(-\frac{1}{2}(x - x_j)^2)} y_i \\ &= \sum_{i=1}^n \text{softmax}(-\frac{1}{2}(x - x_i)^2) y_i\end{aligned}\tag{1}$$

Attention Scoring Functions

ATTENTION POOLING: DIGEST I

- ▶ Re-consider (1):

$$\hat{f}(x) = \sum_{i=1}^n \alpha(x, x_i) y_i = \sum_{i=1}^n \text{softmax}\left(-\frac{1}{2}(x - x_i)^2\right) y_i$$

- ▶ One can view $\alpha(x, x_i)$ as
 - ▶ an attention scoring function

$$a(x, x_i) := -\frac{1}{2}(x - x_i)^2 \quad (2)$$

- ▶ that is further fed into a softmax operation, yielding

$$\alpha(x, x_i) = \text{softmax}(a(x, x_i)) \quad (3)$$

ATTENTION POOLING: DIGEST II

- ▶ One can view $\alpha(x, x_i)$ as
 - ▶ an attention scoring function

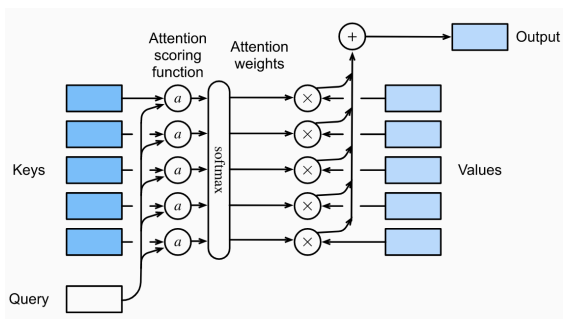
$$a(x, x_i) := -\frac{1}{2}(x - x_i)^2 \quad (4)$$

- ▶ that is further fed into a softmax operation, yielding

$$\alpha(x, x_i) = \text{softmax}(a(x, x_i)) \quad (5)$$

- ▶ *Result:* Probability distribution
 - ▶ over values y_i paired with keys x_i where
 - ▶ probabilities are attention weights $\alpha(x, x_i)$

ATTENTION SCORING FUNCTIONS: MOTIVATION



Output of attention pooling is weighted average of values

- Let x be query, and x_i keys. Attention weights generally compute as

$$\alpha(x, x_i) = \text{softmax}(a(x, x_i)) \quad (6)$$

- *Advantage:* Freedom in choosing attention scoring functions $a(x, x_i)$

ATTENTION POOLING: FORMAL SUMMARY

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

$$f(\mathbf{q}, (\mathbf{k}_1, \mathbf{v}_1), \dots, (\mathbf{k}_m, \mathbf{v}_m)) = \sum_{i=1}^m \alpha(\mathbf{q}, \mathbf{k}_i) \mathbf{v}_i \in \mathbb{R}^v \quad (7)$$

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

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \text{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^m \exp(a(\mathbf{q}, \mathbf{k}_j))} \quad (8)$$

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

$$a : \mathbb{R}^q \times \mathbb{R}^k \longrightarrow \mathbb{R} \quad (9)$$

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

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{w}_v^T \tanh(\mathbf{W}_q \mathbf{q} + \mathbf{W}_k \mathbf{k}) \in \mathbb{R} \quad (10)$$

- ▶ *Interpretation:* (10) 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

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

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k} / \sqrt{d} \quad (11)$$

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

Minibatches:

- ▶ Computing attention for n queries and m keys at once
- ▶ 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

$$\text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V} \in \mathbb{R}^{n \times v} \quad (12)$$

Multi-Head Attention

MULTI-HEAD ATTENTION I

- ▶ *Motivation:* Capture different attention mechanisms for same queries, keys, values
- ▶ *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

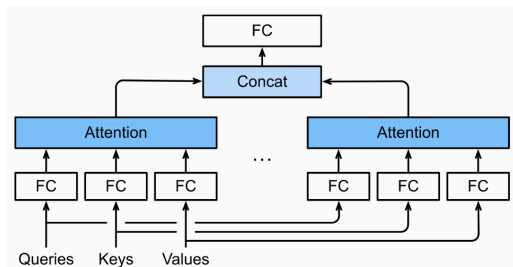
- ▶ *Question:* How to vary attention mechanisms in informed way?
- ▶ *Solution:*
 - ▶ Let h be intended number of attention mechanisms
 - ▶ Linearly transform queries, keys, values using h different sets of matrices $\mathbf{W}_i^{(q)}, \mathbf{W}_i^{(k)}, \mathbf{W}_i^{(v)}, i = 1, \dots, h$
 - ▶ Run the h differently transformed queries, keys, values through attention pooling
 - ▶ Transformations $\mathbf{W}_i^{(q)}, \mathbf{W}_i^{(k)}, \mathbf{W}_i^{(v)}, i = 1, \dots, h$ 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, such as additive (10) or dot-product attention (11)
- ▶ Each attention head is computed as

$$\mathbf{h}_i = f(\mathbf{W}_i^{(q)} \mathbf{q}, \mathbf{W}_i^{(k)} \mathbf{k}, \mathbf{W}_i^{(v)} \mathbf{v}) \in \mathbb{R}^{p_v} \quad (13)$$

MULTI-HEAD ATTENTION IV



From <https://d21.ai>

- ▶ Attention heads:

$$\mathbf{h}_i = f(\mathbf{W}_i^{(q)} \mathbf{q}, \mathbf{W}_i^{(k)} \mathbf{k}, \mathbf{W}_i^{(v)} \mathbf{v}) \in \mathbb{R}^{p_v} \quad (14)$$

- ▶ 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

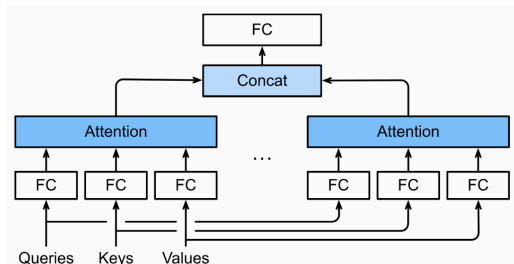
- ▶ Attention heads:

$$\mathbf{h}_i = f(\mathbf{W}_i^{(q)} \mathbf{q}, \mathbf{W}_i^{(k)} \mathbf{k}, \mathbf{W}_i^{(v)} \mathbf{v}) \in \mathbb{R}^{p_v} \quad (15)$$

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

$$\mathbf{W}_o \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_h \end{bmatrix} \in \mathbb{R}^{p_o} \quad (16)$$

MULTI-HEAD ATTENTION II



From <https://d21.ai>

- ▶ Multi-head attention output computes as

$$\mathbf{W}_o[\mathbf{h}_1^T, \dots, \mathbf{h}_h^T]^T \in \mathbb{R}^{p_o} \quad (17)$$

- ▶ 'Concat' layer reflects forming $[\mathbf{h}_1^T, \dots, \mathbf{h}_h^T]$
- ▶ 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

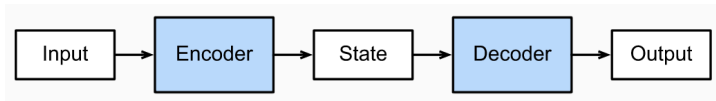
From <https://jalamar.github.io>

SEQUENCE-2-SEQUENCE MODELS III

From <https://jalamar.github.io>

Encoder-Decoder Architecture

ENCODER-DECODER ARCHITECTURE I



From <https://d21.ai>

► *Encoder:*

- Takes input sentence
- Transforms it into *context state*

► *Decoder:*

- Takes context state as input
- Generates output sequence, token by token

ENCODER-DECODER ARCHITECTURE II

From <https://jalammar.github.io>

ENCODER-DECODER ARCHITECTURE III

From <https://jalammar.github.io>

Encoder-Decoder using RNN's

ENCODER-DECODER: RNN REMINDER

Time step #1:

An RNN takes two input vectors:



hidden state #0



input vector #1

Processes them

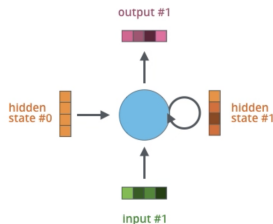
Then produces two output vectors:



hidden state #1



output vector #1



From <https://jalammr.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>

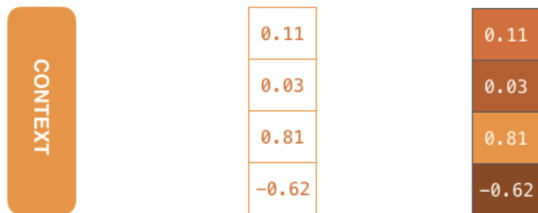
- ▶ *RNN Encoder:*
 - ▶ Uses sentence to translate as input
 - ▶ Generates new hidden state each time step; no output
- ▶ *RNN Decoder:*
 - ▶ Uses last hidden state of encoder as input
 - ▶ Output is translated sentence

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” $\langle \text{eos} \rangle$ token
 - ▶ Decoder stops when generating “end-of-sequence” $\langle \text{eos} \rangle$ token

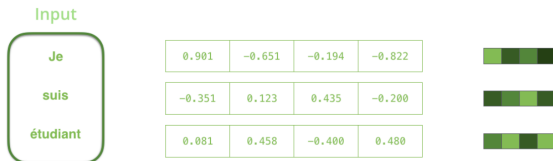
ENCODER-DECODER: CONTEXT



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://jalamar.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, \dots, 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 :

$$f(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad (18)$$

where f expresses the transformation of the encoder's recurrent layer

- ▶ In general, the context variable \mathbf{c} is computed as

$$\mathbf{c} = q(\mathbf{h}_1, \dots, \mathbf{h}_T) \quad (19)$$

where q is a customized function

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

RNN DECODER: FORMAL DESCRIPTION

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

$$\mathbf{P}(y_{t'+1} \mid y_1, \dots, y_{t'}, \mathbf{c}) \quad (20)$$

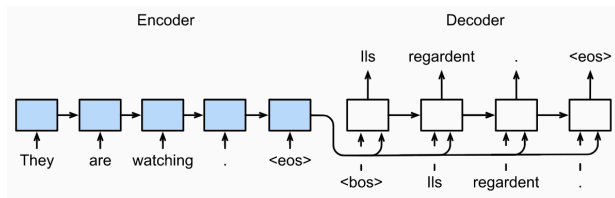
for all possible $y_{t'+1}$

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

$$\mathbf{s}_{t'} = g(y_{t'}, \mathbf{c}, \mathbf{s}_{t'-1}) \quad (21)$$

- ▶ Given $\mathbf{s}_{t'}$, one uses output layer and softmax operation to compute (20)

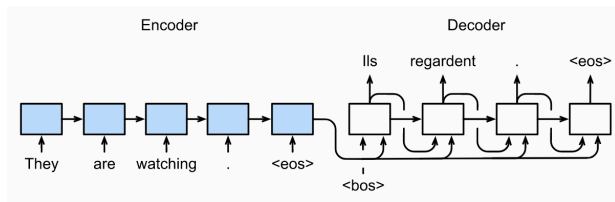
TRAINING: TEACHER FORCING



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
 - ▶ < bos > and < 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 $\langle \text{bos} \rangle$ as input
- ▶ *Simple strategy*: Predict $y_{t'+1}$ that maximizes $\mathbf{P}(y_{t'+1} \mid y_1, \dots, y_{t'}, \mathbf{c})$
 - ▶ *Beware*: Resulting $y_1, \dots, y_{T'}$ may not maximize $\mathbf{P}(y_1, \dots, y_{T'})$
 - ▶ More complex strategies available (e.g. <https://d2l.ai>, 10.8)
- ▶ When $\langle \text{eos} \rangle$ 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

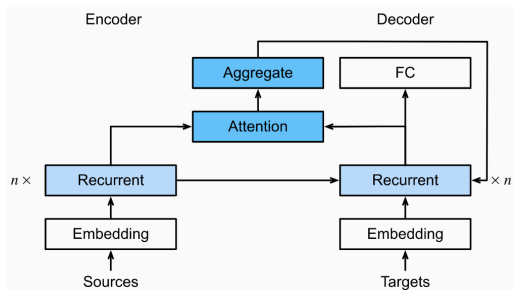
- ▶ Let \mathbf{h}_t be the hidden state of the encoder at time t
- ▶ Let $\mathbf{s}_{t'-1}$ be the hidden state of the decoder at time $t' - 1$
- ▶ Let $\mathbf{c}_{t'}$ be the context variable (i.e. state) after time t'
- ▶ Taking $\mathbf{s}_{t'-1}$ as query, and \mathbf{h}_t as both key and value

$$\mathbf{c}_{t'} = \sum_{t=1}^T \alpha(\mathbf{s}_{t'-1}, \mathbf{h}_t) \mathbf{h}_t \quad (22)$$

determines $\mathbf{c}_{t'}$ where T is the length of the input sequence

- ▶ α reflects the additive attention scoring function (10)
- ▶ One further proceeds using formulas (20),(21)

BAHDANAU ATTENTION: SCHEMATIC



Schematic of Bahdanau Attention

From <https://d21.ai>

- One can integrate already generated tokens into attention (22): see <https://arxiv.org/pdf/1508.01211.pdf>

Self-Attention

SELF-ATTENTION: DEFINITION

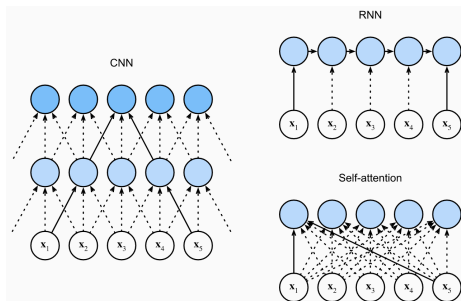
- ▶ Consider a sequence of tokens $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$
- ▶ Each token has its own query, key, and value
- ▶ Hence, each token can attend to each other token:
 - ▶ Pair the query vector with the key of the other token
 - ▶ This yields a weight for its own value
- ▶ Compute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

SELF-ATTENTION: FORMAL SUMMARY

- ▶ Consider a sequence of tokens $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$
- ▶ Replace \mathbf{q} with \mathbf{x} and both $\mathbf{k}_i, \mathbf{v}_i$ with \mathbf{x}_i in (7)
- ▶ One obtains a new sequence $\mathbf{y}_1, \dots, \mathbf{y}_n \in \mathbb{R}^d$ by

$$\mathbf{y}_i := f(\mathbf{x}_i, ((\mathbf{x}_1, \mathbf{x}_1), \dots, (\mathbf{x}_n, \mathbf{x}_n))) = \sum_{j=1}^n \alpha(\mathbf{x}_i, \mathbf{x}_j) \mathbf{x}_j \in \mathbb{R}^d \quad (23)$$

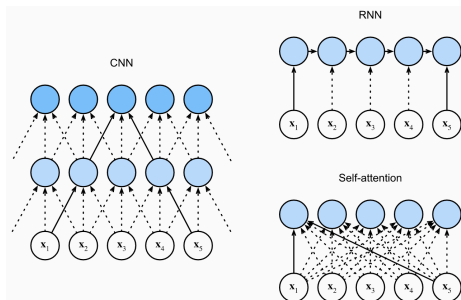
COMPUTATIONAL COMPLEXITY: COMPARISON I



From <https://d21.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

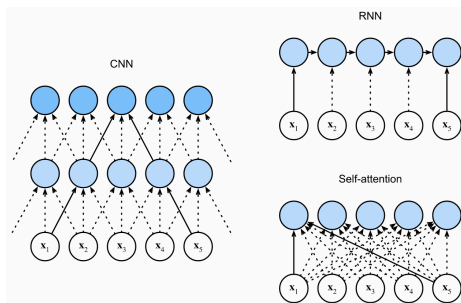
COMPUTATIONAL COMPLEXITY: COMPARISON II



From <https://d21.ai>

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

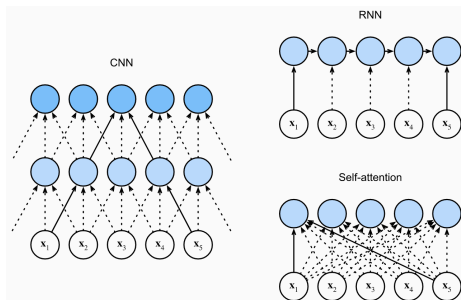
COMPUTATIONAL COMPLEXITY: COMPARISON III



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

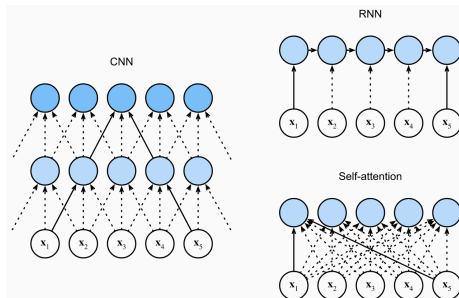
COMPUTATIONAL COMPLEXITY: CNN'S



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:* $\mathcal{O}(n/k)$

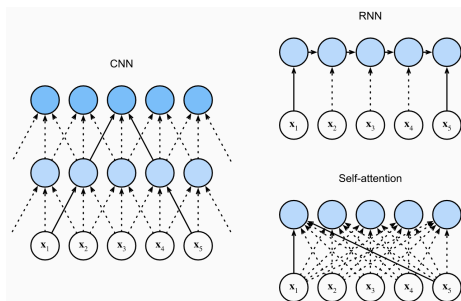
COMPUTATIONAL COMPLEXITY: RNN'S



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://d21.ai>

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

REFERENCES

- ▶ Jay Alammar's ML blog: "Visualizing A Neural Machine Translation Model", see <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>
- ▶ <http://d2l.ai>, 11.4, 11.5, 11.6

OUTLOOK

- ▶ Transformers
- ▶ Phyloformer: Reconstructing phylogenetic trees
- ▶ AlphaFold2: Predicting protein structure from sequence

Thanks for your attention