# Natural Language Processing with Deep Learning

## Lecture 8 – Text generation 2: Autoregressive encoder-decoder with RNNs and attention


Prof. Dr. Ivan Habernal


November 31, 2023


Natural Language Processing Group
Paderborn University
We focus on Trustworthy Human Language Technologies

PADERBORN UNIVERSITY

www.trusthlt.org

# Motivation

Language data – working with sequences (of tokens, characters, etc.)

MLP – fixed input sequence length ❌

RNN – variable length of **input** sequence ✔

# Motivation

Language data – working with sequences (of tokens, characters, etc.)

MLP – fixed input sequence length ❌

RNN – variable length of **input** sequence ✓

What about variable lengths of **output** sequences (compared to input)?

- Text classification ✓
- Sequence labeling ✓
- Sequence generation: translation, summarization 🤔

# Overview of NLP tasks

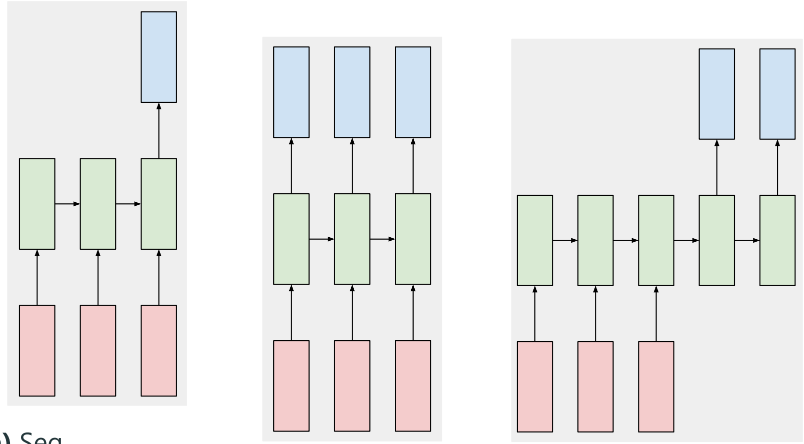

Overview of NLP tasks

Encoder-decoder architectures

The attention mechanism

- Abstracted attention mechanism
- The attention mechanism: design choices

# Overview of NLP tasks

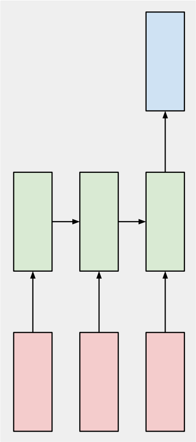


**(a)** Seq. classification

**(b)** Seq. labeling

**(c)** Sequence-to-sequence

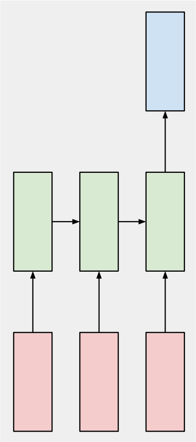
# Sequence classification



Determine a label for one (or more) text sequences

- News article categorization, sentiment analysis,...

# Sequence classification



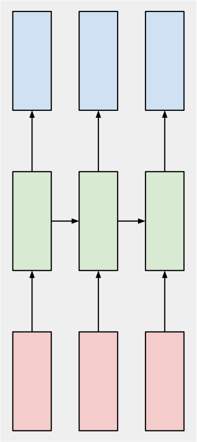
Determine a label for one (or more) text sequences

- News article categorization, sentiment analysis,...

**Approach**

1. Encode sequence(s) into a sequence representation
2. Pass sequence representation to decoder layer

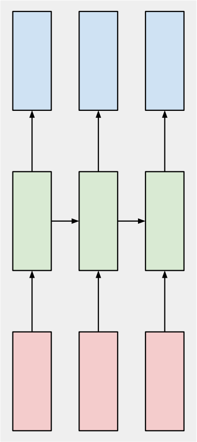
# Sequence labeling



Determine a label for **each element** of a sequence

- Part-of-speech tagging, named entity recognition,...

# Sequence labeling



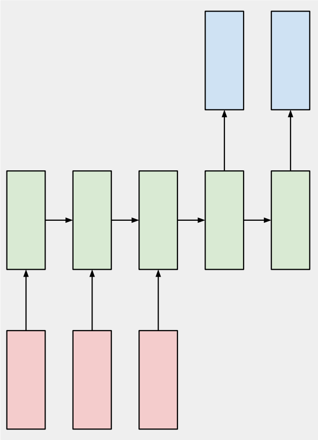
Determine a label for **each element** of a sequence

- Part-of-speech tagging, named entity recognition,...

**Approach**

1. Encode (contextualize) sequence elements
2. Pass representation of each element to (same) decoder layer

# Sequence to sequence



Generate a sequence of tokens given a sequence of tokens

- Machine translation, summarization, text generation,...

**Approach**

1. Use encoder network to encode input sequence
2. Use decoder network to generate output sequence

# Encoder-decoder architectures

Overview of NLP tasks
Encoder-decoder architectures
The attention mechanism
    Abstracted attention mechanism
    The attention mechanism: design choices

# The problem of variable output sequence length

We have a sequence of $n$ **input** vectors $\boldsymbol{x_{1:n}} = \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$

Each input vector has the same dimension $d_{in}$ : $\boldsymbol{x_i} \in \mathbb{R}^{d_{in}}$

# The problem of variable output sequence length

We have a sequence of $n$ **input** vectors $\boldsymbol{x_{1:n}} = \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$

Each input vector has the same dimension $d_{in}$ : $\boldsymbol{x_i} \in \mathbb{R}^{d_{in}}$

We also have a **sequence** of $d_{out}$-dimensional vector $\boldsymbol{y_{1:\hat{n}}} \in \mathbb{R}^{\hat{n} \times d_{out}}$ **outputs**

# The problem of variable output sequence length

We have a sequence of $n$ **input** vectors $\boldsymbol{x_{1:n}} = \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$

Each input vector has the same dimension $d_{in}$ : $\boldsymbol{x_i} \in \mathbb{R}^{d_{in}}$

We also have a **sequence** of $d_{out}$-dimensional vector $\boldsymbol{y_{1:\hat{n}}} \in \mathbb{R}^{\hat{n} \times d_{out}}$ **outputs**

RNNs produce a sequence of outputs

$$\boldsymbol{y_{1:n}} = \mathrm{RNN}(\boldsymbol{x_{1:n}})$$

- **What are we missing?**

# The problem of variable output sequence length

We have a sequence of $n$ **input** vectors $\boldsymbol{x_{1:n}} = \boldsymbol{x_1}, \dots, \boldsymbol{x_n}$

Each input vector has the same dimension $d_{in}$ : $\boldsymbol{x_i} \in \mathbb{R}^{d_{in}}$

We also have a **sequence** of $d_{out}$-dimensional vector $\boldsymbol{y_{1:\hat{n}}} \in \mathbb{R}^{\hat{n} \times d_{out}}$ **outputs**

RNNs produce a sequence of outputs

$$\boldsymbol{y_{1:n}} = \mathrm{RNN}(\boldsymbol{x_{1:n}})$$

- **What are we missing?**
  - The input and output sequence are rarely of same length

# Generating a variable length sequence

**Translate to German**: *I like attending deep learning lectures*

# Generating a variable length sequence

**Translate to German**: *I like attending deep learning lectures*

**Output**: *Ich besuche gerne Deep-Learning-Vorlesungen*

# Generating a variable length sequence

**Translate to German**: *I like attending deep learning lectures*

**Output**: *Ich besuche gerne Deep-Learning-Vorlesungen*

Current approach:

1. Tokenize input sequence
2. Obtain a word embedding (e.g. word2vec) for each token
3. Use a RNN (e.g. LSTM) to encode sequence of tokens
4. Generate token sequence in target language

# Generating a variable length sequence

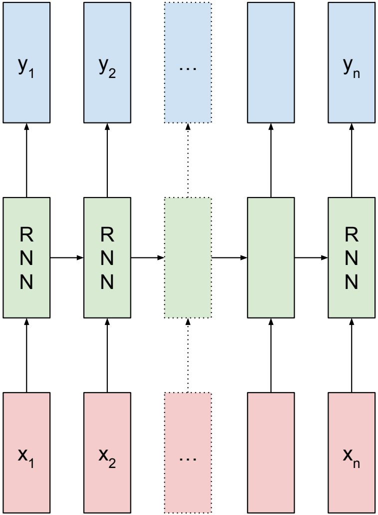
**Translate to German**: *I like attending deep learning lectures*

**Output**: *Ich besuche gerne Deep-Learning-Vorlesungen*

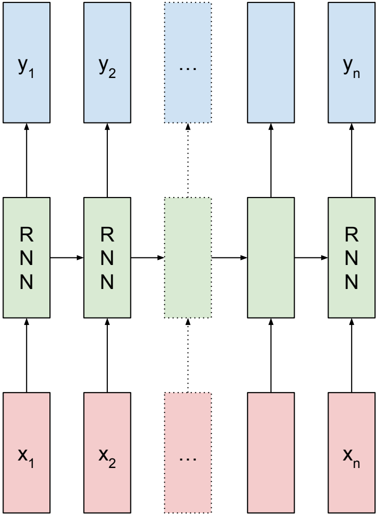
Current approach:

1. Tokenize input sequence
2. Obtain a word embedding (e.g. word2vec) for each token
3. Use a RNN (e.g. LSTM) to encode sequence of tokens
4. Generate token sequence in target language
   - Multi-class classification over target vocabulary

# Generating a variable length sequence

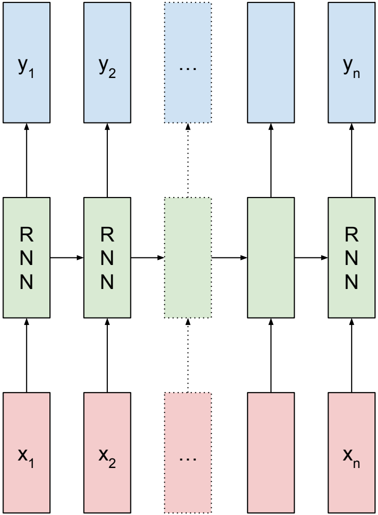


# Generating a variable length sequence




How to solve the issue of varying input/output lengths?

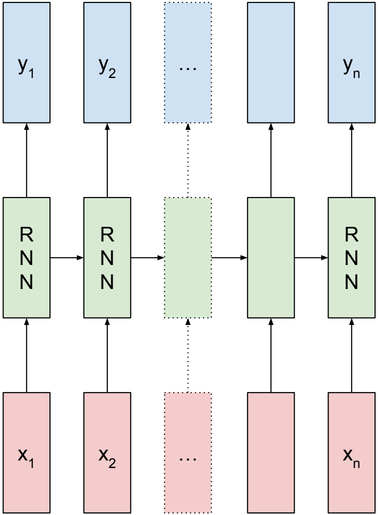
# Generating a variable length sequence




How to solve the issue of varying input/output lengths?

1. We **don't have to** stop generating after the last input
2. We can only consider outputs up to a special **"end token"**

# Generating a variable length sequence

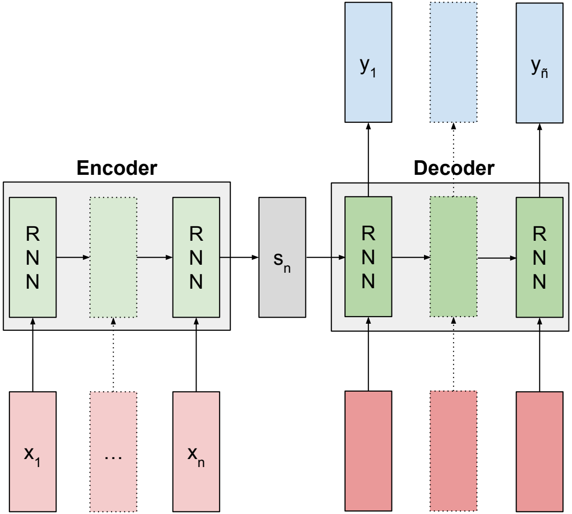



How to solve the issue of varying input/output lengths?

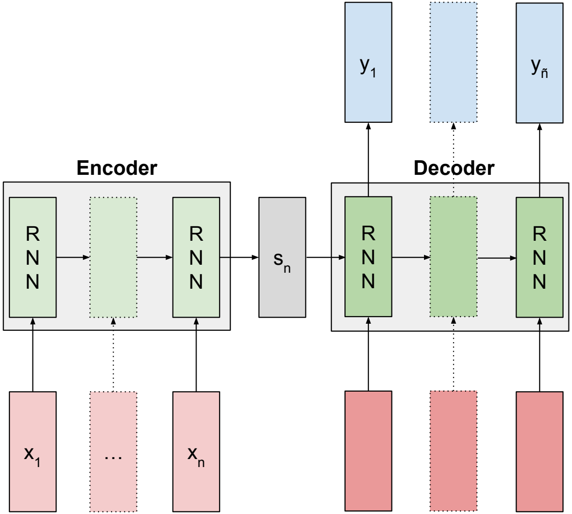
1. We **don't have to** stop generating after the last input
2. We can only consider outputs up to a special **"end token"**

Neither ideal

# Sequence-to-sequence models



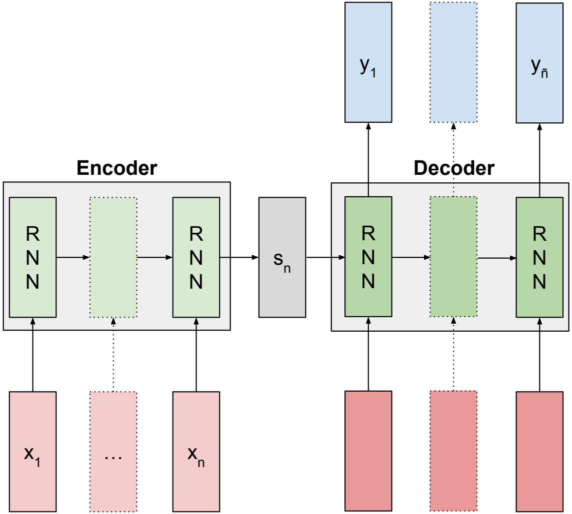
# Sequence-to-sequence models




Two networks

- **Encoder** (reader) RNN
- **Decoder** (writer) RNN

# Sequence-to-sequence models




Two networks

- **Encoder** (reader) RNN
- **Decoder** (writer) RNN

Note:

- Encoder and decoder have **separate** params

# The encoder-decoder architecture specifics

1. How to **initialize** decoder hidden **state**?

# The encoder-decoder architecture specifics

1. How to **initialize** decoder hidden **state**?
    - $h_0^{dec} = h_n^{enc}$: simply copy the last encoder state
    - $h_0^{dec} = \text{NN}_\theta(h_n^{enc})$: transform the last encoder state

# The encoder-decoder architecture specifics

1. How to **initialize** decoder hidden **state**?
   - $h_0^{dec} = h_n^{enc}$: simply copy the last encoder state
   - $h_0^{dec} = \text{NN}_\theta(h_n^{enc})$: transform the last encoder state
2. When do we **stop generating** with the decoder?

# The encoder-decoder architecture specifics

1. How to **initialize** decoder hidden **state**?
    - $h_0^{dec} = h_n^{enc}$: simply copy the last encoder state
    - $h_0^{dec} = \text{NN}_\theta(h_n^{enc})$: transform the last encoder state
2. When do we **stop generating** with the decoder?
    - We use a **special token** (<EOS>, \n) to indicate the end-of-sequence
    - When the **maximum generation length** is exceeded

# The encoder-decoder architecture specifics

1. How to **initialize** decoder hidden **state**?
   - $h_0^{dec} = h_n^{enc}$: simply copy the last encoder state
   - $h_0^{dec} = \text{NN}_\theta(h_n^{enc})$: transform the last encoder state
2. When do we **stop generating** with the decoder?
   - We use a **special token** (<EOS>, \n) to indicate the end-of-sequence
   - When the **maximum generation length** is exceeded
3. What are the **inputs** of the decoder?

# The encoder-decoder architecture specifics

1. How to **initialize** decoder hidden **state**?
    - $h_0^{dec} = h_n^{enc}$: simply copy the last encoder state
    - $h_0^{dec} = \text{NN}_\theta(h_n^{enc})$: transform the last encoder state
2. When do we **stop generating** with the decoder?
    - We use a **special token** (`<EOS>`, `\n`) to indicate the end-of-sequence
    - When the **maximum generation length** is exceeded
3. What are the **inputs** of the decoder?
    - The **previous output** of the decoder
        - Teacher forcing (with probability $p$): use the **correct output**

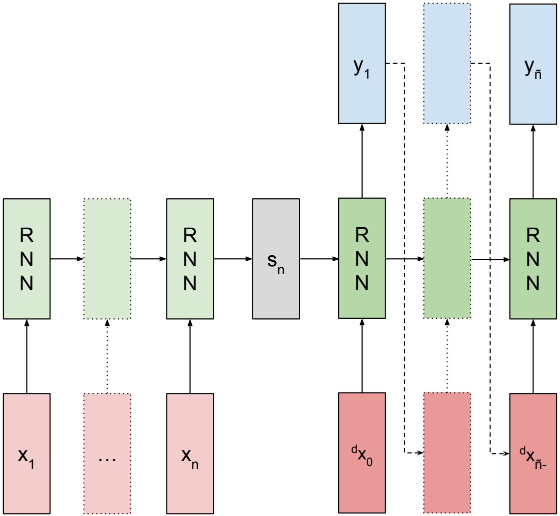
# The encoder-decoder architecture specifics

1. How to **initialize** decoder hidden **state**?
    - $h_0^{dec} = h_n^{enc}$: simply copy the last encoder state
    - $h_0^{dec} = \text{NN}_\theta(h_n^{enc})$: transform the last encoder state
2. When do we **stop generating** with the decoder?
    - We use a **special token** (<EOS>, \n) to indicate the end-of-sequence
    - When the **maximum generation length** is exceeded
3. What are the **inputs** of the decoder?
    - The **previous output** of the decoder
        - Teacher forcing (with probability $p$): use the **correct output**
    - What is the **initial input** $x_0^{dec}$?

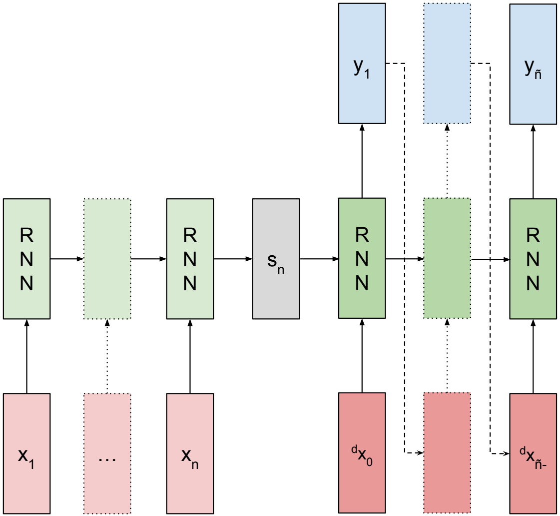
# The encoder-decoder architecture specifics

1. How to **initialize** decoder hidden **state**?
   - $h_0^{dec} = h_n^{enc}$: simply copy the last encoder state
   - $h_0^{dec} = \text{NN}_\theta(h_n^{enc})$: transform the last encoder state
2. When do we **stop generating** with the decoder?
   - We use a **special token** (<EOS>, \n) to indicate the end-of-sequence
   - When the **maximum generation length** is exceeded
3. What are the **inputs** of the decoder?
   - The **previous output** of the decoder
     - Teacher forcing (with probability $p$): use the **correct output**
   - What is the **initial input** $x_0^{dec}$?
     - A beginning-of-sequence **special token** (<BOS>)

# The encoder-decoder architecture



# The encoder-decoder architecture




Decoder inputs

- $x_0^{dec}$ = <BOS>
- $x_i^{dec} = y_i^{dec}$ **if** no teacher forcing
- $x_i^{dec} = \hat{y}_i$ **if** we use teacher forcing

# Summary

- Sequence generation tasks difficult to solve with a single RNN
- Encoder-decoder architecture: use two separate RNN networks
  - The encoder reads the input text and compresses it into a fixed size vector
  - The decoder uses the input text representation and generates output text
- Encoder-decoder specifics:
  - Special tokens: <BOS>, <EOS>
  - Helping the network: teacher forcing

# The attention mechanism

Overview of NLP tasks

Encoder-decoder architectures

The attention mechanism

- Abstracted attention mechanism
- The attention mechanism: design choices

# Motivation

… we apply our multilayer bidirectional LSTM network to a machine translation problem.

What **types of instances** would it perform bad on? **Why**?

# Motivation

… we apply our multilayer bidirectional LSTM network to a machine translation problem.

What **types of instances** would it perform bad on? **Why**?

The problem of **long dependencies**

- The hidden state of a RNN network is **finite**
- The more tokens the RNN reads, the less it remembers **individual** tokens

# The long dependency problem

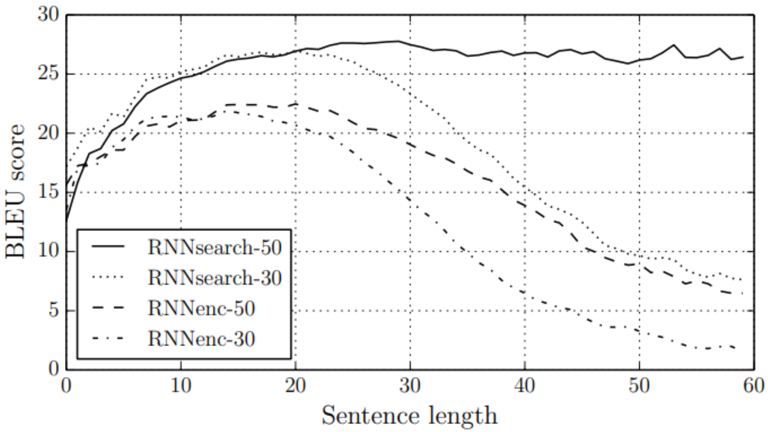



Figure from Bahdanau, Cho, and Bengio, 2015

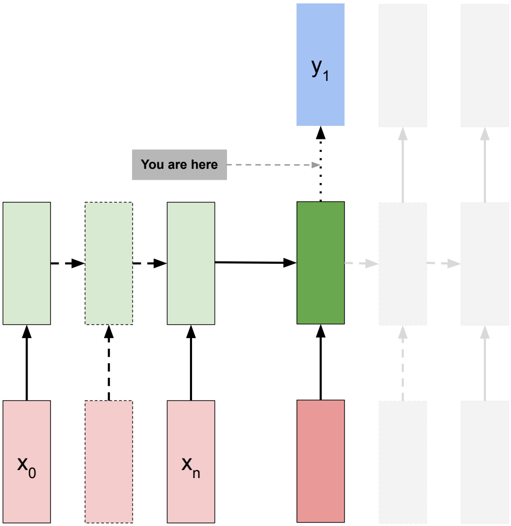
D. Bahdanau, K. Cho, and Y. Bengio (2015). **"Neural Machine Translation by Jointly Learning to Align and Translate".** In: *3rd International Conference on Learning Representations (ICLR)*. ed. by Y. Bengio and Y. LeCun. San Diego, CA, USA

# The attention mechanism: intuition

**Idea**: our recurrent state does not have perfect memory of previous content. However, it should know which content **was relevant**.

- Attention allows the network to **view previous states**

# The attention mechanism: visual context



# The attention mechanism: formalization

A standard encoder-decoder network produces a **sequence of states** $s_i^{\text{enc/dec}}$

# The attention mechanism: formalization

A standard encoder-decoder network produces a **sequence of states** $s_i^{\text{enc/dec}}$

At a (decoder) time-step $t$, we want to obtain a **fixed size** update (with respect to sequence length $N$) representing **relevant information** from the past

# The attention mechanism: formalization

A standard encoder-decoder network produces a **sequence of states** $s_i^{\text{enc/dec}}$

At a (decoder) time-step $t$, we want to obtain a **fixed size** update (with respect to sequence length $N$) representing **relevant information** from the past

We **have**: $s_t^{\text{dec}}$, $S^{\text{enc}} = \{s_i^{\text{enc}}\}_{i=1}^{n}$, we **want**:
$a \approx \text{relevant}(S^{\text{enc}} | s_t^{\text{dec}})$

# The attention mechanism: formalization

1. Compute the **energy** (similarity, relevance) function between two dense vectors (the **current** decoder state and **one** encoder state)

$$\alpha_i = \mathsf{attn}(s_i^{\mathsf{enc}}, s_t^{\mathsf{dec}})$$

# The attention mechanism: formalization

1. Compute the **energy** (similarity, relevance) function between two dense vectors (the **current** decoder state and **one** encoder state)

$$\alpha_i = \text{attn}(s_i^{\text{enc}}, s_t^{\text{dec}}) \approx \underbrace{s_i^{\text{enc}} \cdot s_t^{\text{dec}}}_{\text{dot product}}$$

# The attention mechanism: formalization

2. We **scale** the output of the dot product to preserve scale of variance (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, and Polosukhin, 2017) (otherwise values get too large – issue for next step)

$$\hat{\alpha}_i = \frac{s_i^{\text{enc}} \cdot s_t^{\text{dec}}}{\sqrt{d_{\text{dec}}}}$$

# The attention mechanism: formalization

2. We **scale** the output of the dot product to preserve scale of variance (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, and Polosukhin, 2017) (otherwise values get too large – issue for next step)

$$\hat{\alpha}_i = \frac{s_i^{\text{enc}} \cdot s_t^{\text{dec}}}{\sqrt{d_{\text{dec}}}}$$

$d_{\text{dec}}$ is the dimensionality of the decoder state

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). **"Attention Is All You Need".** In: *Advances in Neural Information Processing Systems 30*. Long Beach, CA, USA: Curran Associates, Inc., pp. 5998–6008

# The attention mechanism: formalization

3. We **normalize** the energy to a probability distribution over (encoder) states

$$\alpha_i = \mathrm{softmax}(\hat{\alpha}_i) = \frac{\exp(\hat{\alpha}_i)}{\sum_j^N \exp(\hat{\alpha}_j)}$$

# The attention mechanism: formalization

3. We **normalize** the energy to a probability distribution over (encoder) states

$$\alpha_i = \mathrm{softmax}(\hat{\alpha}_i) = \frac{\exp(\hat{\alpha}_i)}{\sum_j^N \exp(\hat{\alpha}_j)}$$

Why would the scale of $\hat{\alpha}_i$ be an issue?

# The attention mechanism: formalization

3. We **normalize** the energy to a probability distribution over (encoder) states

$$\alpha_i = \mathrm{softmax}(\hat{\alpha}_i) = \frac{\exp(\hat{\alpha}_i)}{\sum_j^N \exp(\hat{\alpha}_j)}$$

Why would the scale of $\hat{\alpha}_i$ be an issue? (softmax)

# The attention mechanism: formalization

We now have **importance** $\alpha_i$ of each (encoder) element.
How to produce the *summary* $a$?

4. We can **sum** over the elements with $\alpha_i$ as the elements' weight!

$$a = \sum_{i}^{n} \alpha_i s_i^{\text{enc}}$$

# The attention mechanism: formalization

We now have **importance** $\alpha_i$ of each (encoder) element.

How to produce the *summary* $a$?

4. We can **sum** over the elements with $\alpha_i$ as the elements' weight!

$$a = \sum_{i}^{n} \alpha_i s_i^{\text{enc}}$$

- $\alpha_i \approx$ **importance** of state $s_i^{\text{enc}}$
- $s_i^{\text{enc}} \approx$ information we are **recalling**

# The attention mechanism: formalization

We now have **importance** $\alpha_i$ of each (encoder) element.
How to produce the *summary* $a$?

4. We can **sum** over the elements with $\alpha_i$ as the elements' weight!

$$a = \sum_i^n \alpha_i s_i^{\text{enc}}$$

- $\alpha_i \approx$ **importance** of state $s_i^{\text{enc}}$
- $s_i^{\text{enc}} \approx$ information we are **recalling**

This is the initial formulation of **dot-product encoder-decoder** attention.

# The attention mechanism: formalization

We now have **importance** $\alpha_i$ of each (encoder) element.
How to produce the *summary* $a$?

4. We can **sum** over the elements with $\alpha_i$ as the elements' weight!

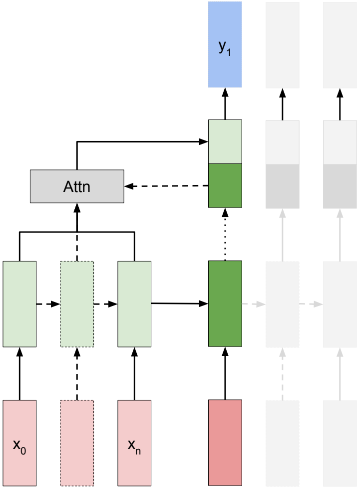
$$a = \sum_{i}^{n} \alpha_i s_i^{\text{enc}}$$

- $\alpha_i \approx$ **importance** of state $s_i^{\text{enc}}$
- $s_i^{\text{enc}} \approx$ information we are **recalling**

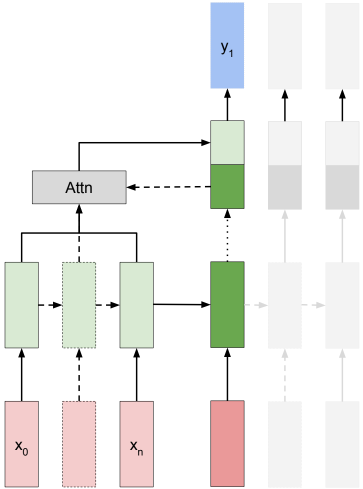
This is the initial formulation of **dot-product encoder-decoder** attention.

- Attention is a weighted sum over a set of elements.

# The attention mechanism: visual context

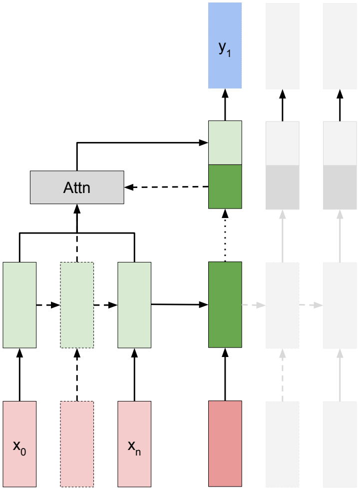


# The attention mechanism: visual context



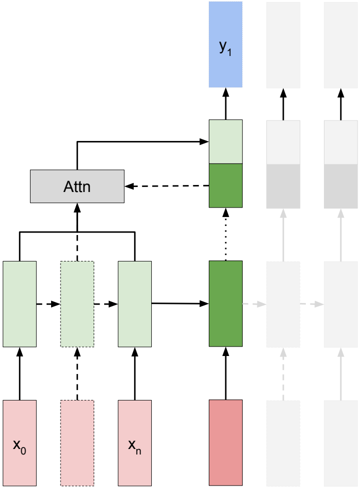

1. Given the current decoder state $s_t^{\text{dec}}$ and encoder states $S^{\text{enc}} = \{s_i^{\text{enc}}\}_{i=1}^{n}$ compute the output of the attention mechanism $a_t = \sum_i^n \alpha_i s_i^{\text{enc}}$

# The attention mechanism: visual context



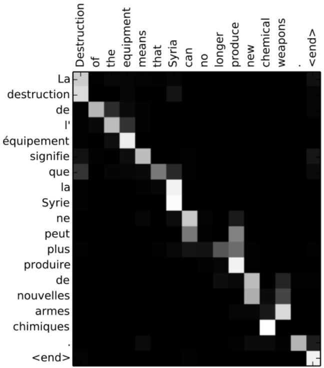

1. Given the current decoder state $s_t^{\text{dec}}$ and encoder states $S^{\text{enc}} = \{s_i^{\text{enc}}\}_{i=1}^{n}$ compute the output of the attention mechanism $a_t = \sum_i^n \alpha_i s_i^{\text{enc}}$
2. **Concatenate** the output of attention and the current decoder state $\hat{s}_t^{\text{dec}} = [s_t^{\text{dec}} | a_t]$

# The attention mechanism: visual context




1. Given the current decoder state $s_t^{\text{dec}}$ and encoder states $S^{\text{enc}} = \{s_i^{\text{enc}}\}_{i=1}^n$ compute the output of the attention mechanism $a_t = \sum_i^n \alpha_i s_i^{\text{enc}}$
2. **Concatenate** the output of attention and the current decoder state $\hat{s}_t^{\text{dec}} = [s_t^{\text{dec}} | a_t]$
3. Predict the next token $y_t$

# The attention mechanism: visualizing attention

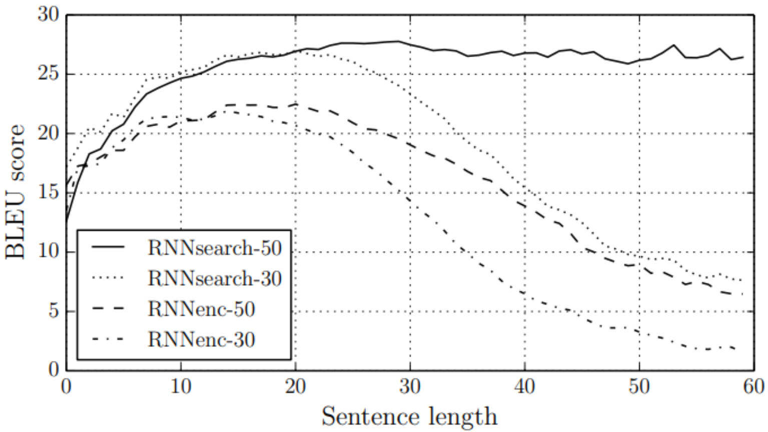



D. Bahdanau, K. Cho, and Y. Bengio (2015). **"Neural Machine Translation by Jointly Learning to Align and Translate".** In: *3rd International Conference on Learning Representations (ICLR)*. ed. by Y. Bengio and Y. LeCun. San Diego, CA, USA

White = attention (btw enc. and dec. state) is high. Black = attention is low.

Figure from Bahdanau, Cho, and Bengio, 2015.

# The attention mechanism: effect of attention




D. Bahdanau, K. Cho, and Y. Bengio (2015). **"Neural Machine Translation by Jointly Learning to Align and Translate".** In: *3rd International Conference on Learning Representations (ICLR)*. ed. by Y. Bengio and Y. LeCun. San Diego, CA, USA

RNNsearch architectures use attention. Figure from Bahdanau, Cho, and Bengio, 2015.

# The attention mechanism

## Abstracted attention mechanism

# The attention mechanism: abstraction (Components)

1. The **query** $q = f_q(s_t^{\text{dec}}); \quad q \in \mathbb{R}^{d_q}$
    - The query is the state representation based on which we **seek information**

# The attention mechanism: abstraction (Components)

1. The **query** $q = f_q(s_t^{\text{dec}}); \quad q \in \mathbb{R}^{d_q}$
    - The query is the state representation based on which we **seek information**
2. The **keys** $K = f_k(\{s_i^{\text{enc}}\}_{i=1}^{n}); \quad K \in \mathbb{R}^{n \times d_k}$
    - The keys are the representations we **compare** the query to

# The attention mechanism: abstraction (Components)

1. The **query** $q = f_q(s_t^{\text{dec}}); \quad q \in \mathbb{R}^{d_q}$
   - The query is the state representation based on which we **seek information**
2. The **keys** $K = f_k(\{s_i^{\text{enc}}\}_{i=1}^n); \quad K \in \mathbb{R}^{n \times d_k}$
   - The keys are the representations we **compare** the query to
3. The **values** $V = f_v(\{s_i^{\text{enc}}\}_{i=1}^n); \quad V \in \mathbb{R}^{n \times d_v}$
   - The values are the representations we **sum over** given the attention scores

# The attention mechanism: abstraction (Components)

1. The **query** $q = f_q(s_t^{\text{dec}}); \quad q \in \mathbb{R}^{d_q}$
   - The query is the state representation based on which we **seek information**
2. The **keys** $K = f_k(\{s_i^{\text{enc}}\}_{i=1}^{n}); \quad K \in \mathbb{R}^{n \times d_k}$
   - The keys are the representations we **compare** the query to
3. The **values** $V = f_v(\{s_i^{\text{enc}}\}_{i=1}^{n}); \quad V \in \mathbb{R}^{n \times d_v}$
   - The values are the representations we **sum over** given the attention scores

Where $f_q, f_k, f_v$ are arbitrary functions (neural network layers).

# The attention mechanism: abstraction (Components)

1. The **query** $q = f_q(s_t^{\text{dec}}); \quad q \in \mathbb{R}^{d_q}$
   - The query is the state representation based on which we **seek information**
2. The **keys** $K = f_k(\{s_i^{\text{enc}}\}_{i=1}^{n}); \quad K \in \mathbb{R}^{n \times d_k}$
   - The keys are the representations we **compare** the query to
3. The **values** $V = f_v(\{s_i^{\text{enc}}\}_{i=1}^{n}); \quad V \in \mathbb{R}^{n \times d_v}$
   - The values are the representations we **sum over** given the attention scores

Where $f_q, f_k, f_v$ are arbitrary functions (neural network layers).

$$a = \sum_i^n \alpha_i v_i \qquad (1) \qquad\qquad \hat{\alpha}_i = \frac{q^T \cdot k_i}{\sqrt{d_{\text{dec}}}} \qquad (2)$$

# The attention mechanism

## The attention mechanism: design choices

# The attention mechanism: choices

Key choices when using the attention mechanism:

1. The **energy** (similarity, relevance) function
   - Defines how we **compute energy** between two state representations
2. Parametrization
   - Determines how (and if) we **apply transformations** to attention components
3. Direction
   - Determines which components we **attend over**

# The attention mechanism: energy

1. The **energy** (similarity, relevance) function

# The attention mechanism: energy

1. The **energy** (similarity, relevance) function
   - Dot product attention

$$\hat{\alpha}_i = \frac{q^T \cdot k_i}{\sqrt{d_k}}$$

# The attention mechanism: energy

1. The **energy** (similarity, relevance) function
    - Dot product attention

$$\hat{\alpha}_i = \frac{q^T \cdot k_i}{\sqrt{d_k}}$$

- Requires $\dim(q) = \dim(k)$
- Introduces **no additional parameters**

# The attention mechanism: energy

1. The **energy** (similarity, relevance) function
    - Bahdanau (**tanh**) attention ($[\cdot|\cdot]$ = concatenate)

$$\hat{\alpha}_i = W_2 \text{tanh}(W_1 [q | k_i])$$

# The attention mechanism: energy

1. The **energy** (similarity, relevance) function
   - Bahdanau (**tanh**) attention ($[\cdot|\cdot]$ = concatenate)

$$\hat{\alpha}_i = W_2 \text{tanh}(W_1 [q | k_i])$$

- **No requirements** on dimensions of inputs (states)
- Additional parameters $W_1 \in \mathbb{R}^{(d_q + d_k) \times h}$, $W_2 \in \mathbb{R}^h$
- $h$ is the dimension of the **attention hidden layer**

# The attention mechanism: energy

1. The **energy** (similarity, relevance) function
   - **Bilinear** attention

$$\alpha_i = q^T W k_i$$

# The attention mechanism: energy

1. The **energy** (similarity, relevance) function
   - **Bilinear** attention

$$\alpha_i = q^T W k_i$$

- **No requirements** on dimensions of states
- Additional parameters $W \in \mathbb{R}^{d_q \times d_k}$

# The attention mechanism: parametrization

2. Parametrizations of inputs & outputs
   Remember: $f_q, f_k, f_v$ are arbitrary functions (neural network layers).

# The attention mechanism: parametrization

2. Parametrizations of inputs & outputs

   Remember: $f_q, f_k, f_v$ are arbitrary functions (neural network layers).

   What are the most common ways to parametrize these functions?

# The attention mechanism: parametrization

2. Parametrizations of inputs & outputs

   Remember: $f_q, f_k, f_v$ are arbitrary functions (neural network layers).

   What are the most common ways to parametrize these functions?

   - Linear transformations: $f_{\{q,k,v\}} \in \mathbb{R}^{d_{\{q,k,v\}\_in} \times d_{\{q,k,v\}}}$

# The attention mechanism: parametrization

2. Parametrizations of inputs & outputs

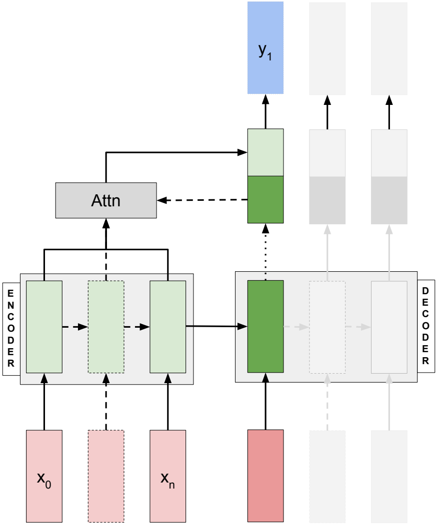
   Remember: $f_q, f_k, f_v$ are arbitrary functions (neural network layers).

   What are the most common ways to parametrize these functions?

   - Linear transformations: $f_{\{q,k,v\}} \in \mathbb{R}^{d_{\{q,k,v\}\_in} \times d_{\{q,k,v\}}}$

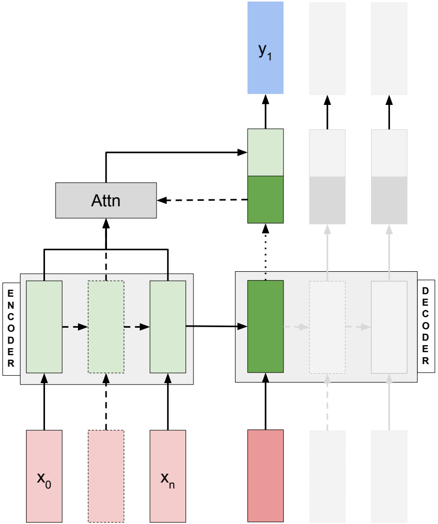
   **Intuition**: hidden states contain information which is not relevant for **computing energy** (query, keys) or **retrieving information** (values) – linear transformations can filter (map to null space) unnecessary information.

# The attention mechanism: direction



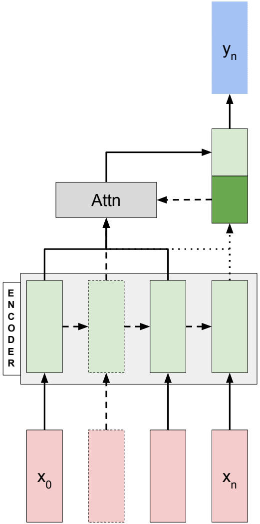

3. Direction of attention
   We have so far only shown encoder-decoder **cross attention**

# The attention mechanism: direction




3. <u>Direction of attention</u>
   We have so far only shown encoder-decoder **cross attention**
   Flavors of attention
   - **Cross-attention**: between encoder and decoder (or any **query** and **a sequence** of hidden states)

# The attention mechanism: direction




3. Direction of attention

   We have so far only shown encoder-decoder **cross attention**

   Flavors of attention

   - **Self-attention**: between a sequence of hidden states and a query originating from the **same sequence** of hidden states

# Recap

Overview of NLP tasks

Encoder-decoder architectures

The attention mechanism

- Abstracted attention mechanism
- The attention mechanism: design choices

# Take aways

- Encoder-decoder architecture – used for generating variable (wrt. input) length sequences
- Three classes of sequence problems: classification, labeling & seq2seq
- RNNs are bad at long dependencies
- Attention mechanism allows networks to *look at* previous states
- Abstraction of attention mechanism: (1) query, (2) keys, (3) values
- Design choices of attention: (1) energy function, (2) parametrization, (3) direction

# License and credits



## Credits

Ivan Habernal, Martin Tutek