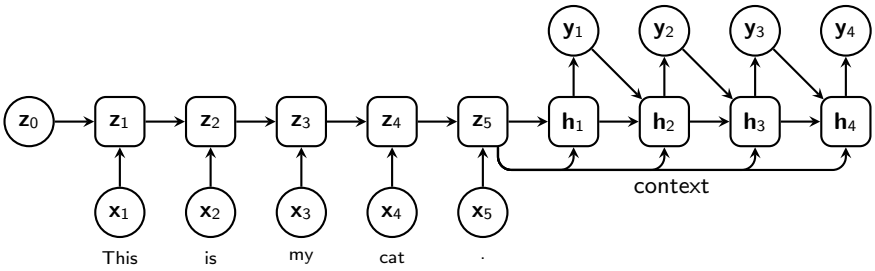
Aalto University

CS-E4890: Deep Learning
Attention-based models

Alexander Ilin

Simple sequence-to-sequence model

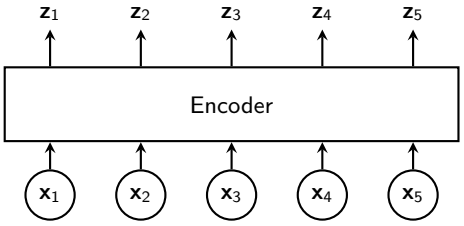
- Previously we considered a sequence-to-sequence model for statistical machine translation:



- The problem with this model: It is difficult to encode the whole sentence in a single vector $\mathbf{z}_5$ of fixed size.

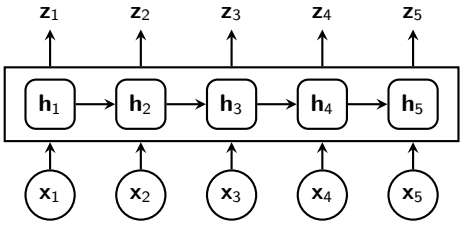
Encoding to representation of a varying-length

- Intuition: The longer the input sentence, the longer our representation should be. Let the length of our representation be equal to the length of the input sequence.



Encoding to representation of a varying-length

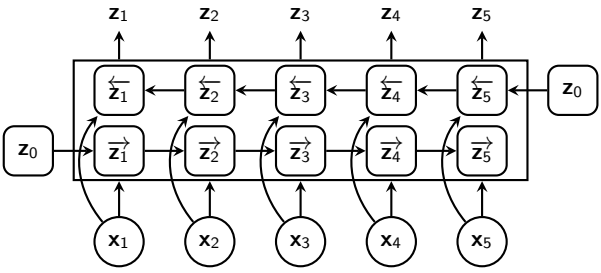
- Intuition: The longer the input sentence, the longer our representation should be. Let the length of our representation be equal to the length of the input sequence.



- We can use intermediate states of the RNN as representations but this does not work well: representation $\mathbf{z}_1$ at the first position does not depend on subsequent words.

Encoding with bi-directional RNN

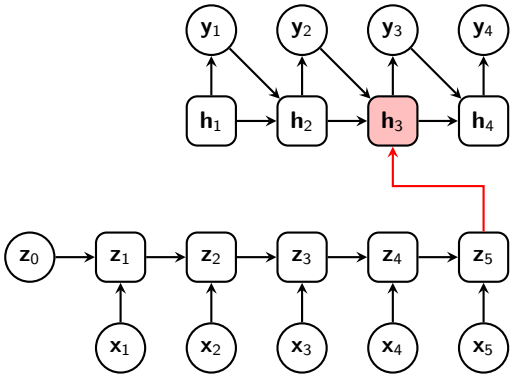
- In the classical model (Bahdanau et al., 2014), the varying-length representation was build using a bi-directional RNN.



- The bi-direction RNN does two passes through the input sequence: forward and backward.
- The output at position j is a concatenation $\mathbf{z}_j = [\overrightarrow{\mathbf{z}_j}; \overleftarrow{\mathbf{z}_j}]$ of the states (or outputs) in the forward and backward passes.

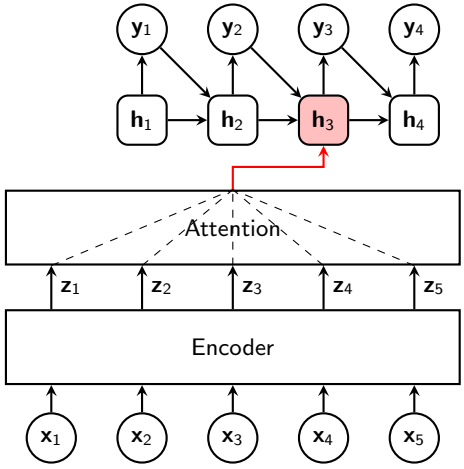
Sequence-to-sequence model: context in decoding

- In the simple seq2seq model (see the first slide), we used the last state of the encoder RNN as the context for decoding in each step.
- What should we do if we want to use an encoded representation of a varying length?



Attention: Using context of varying length

- We can select one of the vectors $\mathbf{z}_j$ as our context when decoding at step i.
- Which one to select? We let the neural network decide it by itself using the *attention* mechanism.
- You can think of attention as a switch that selects one of the inputs $\mathbf{z}_j$.



Attention mechanism from (Bahdanau et al., 2014)

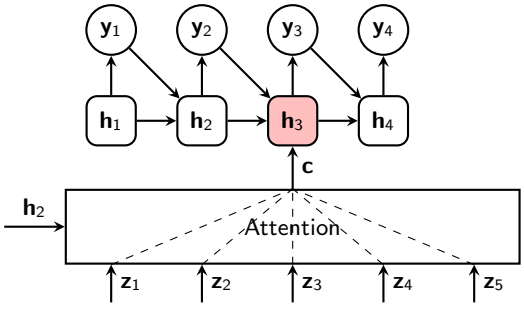
- Select one of the inputs as the output:

$$\mathbf{c} = \sum_{j=1}^{n} \alpha_j \mathbf{z}_j$$

$$0 < \alpha_j < 1, \quad \sum_{j=1}^{n} \alpha_j = 1$$

- Weights α_j are computed using softmax:

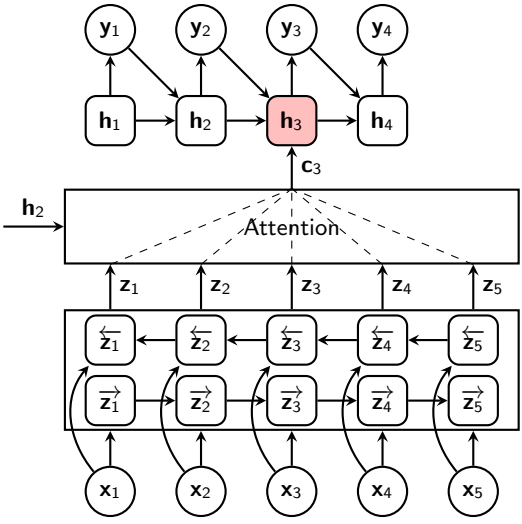
$$\alpha_j = \frac{\exp(e_j)}{\sum_{j'=1}^{n} \exp(e_{j'})}$$



- Scores $e_{ij'}$ are computed using the current decoder state $\mathbf{h}_{i-1}$ and representation $\mathbf{z}_j$:

$$e_j = f(\mathbf{h}_{i-1}, \mathbf{z}_j)$$

Full architecture from (Bahdanau et al., 2014)



Attention-based models have much better performance

- Using attention significantly improves the quality of translation.

The translation performances on an English-to-French translation task (WMT'14)

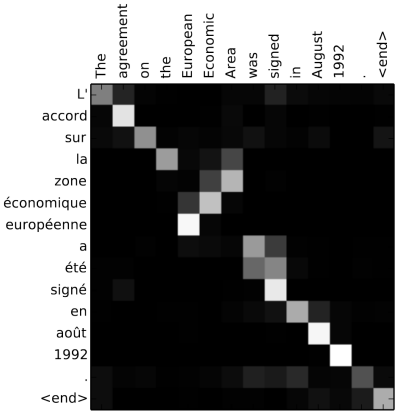
Model	BLEU
Simple Enc-Dec	17.82
Attention-based Enc-Dec	28.45
Attention-based Enc-Dec (LV)	34.11
Attention-based Enc-Dec (LV, ensemble)	37.19

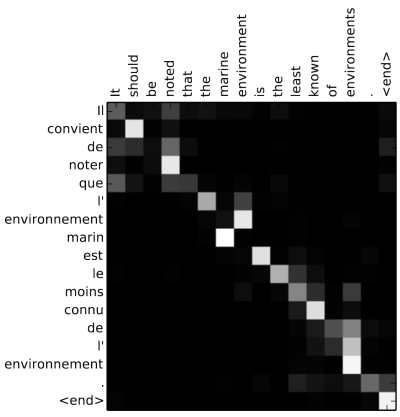
LV - large vocabulary

source: (Jean et al., 2014)

Attention coefficients

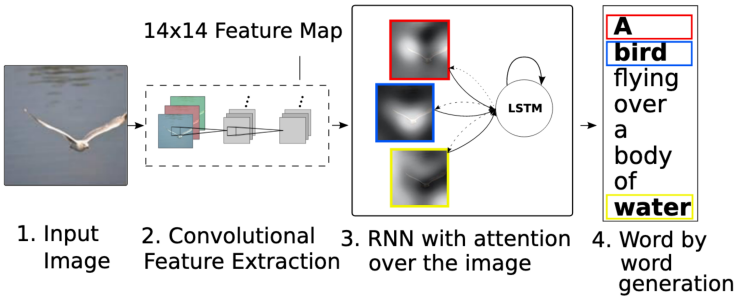
- Weights α_{ij} can be visualized. The x-axis and y-axis of each plot correspond to the words in the source sentence and the generated translation, respectively.





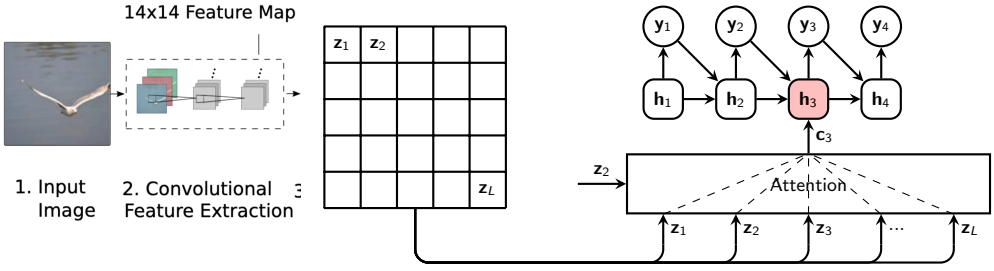
Neural image captioning (Xu et al., 2016)

- Models with attention have been used in many domains.
- "Show, Attend and Tell" paper solves the task of image captionining similarly to a translation task: images are "translated" to sentences.



Neural image captioning (Xu et al., 2016)

- The image is preprocessed into 14×14 feature maps with a convolutional network pre-trained on ImageNet.
- The 14×14 feature maps are split into L annotation vectors $\mathbf{z}_j$.
- The annotation vectors are used as context in the decoding RNN.

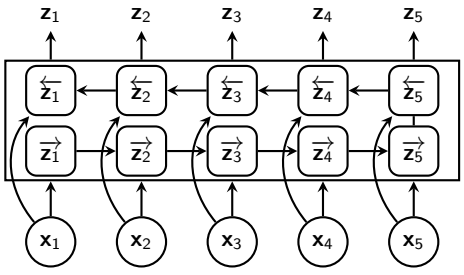


Convolutional sequence-to-sequence models

(Gehring et al., 2017)

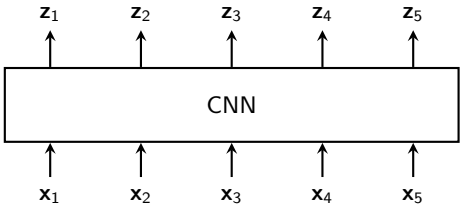
Problems with RNN encoders

- Problem with RNN encoding:
  - The number of steps is equal to the number of words in the input sentence. This can make training slow.
  - We need to take multiple steps to model relations between distant words. Modeling long-term dependencies can be difficult with RNNs.
- Since we know how to deal with encodings of varying lengths (using attention), we do not really need to use an RNN. The encoder can be any network that converts input sequence $(\mathbf{x}_1, ..., \mathbf{x}_n)$ into representations $(\mathbf{z}_1, ..., \mathbf{z}_n)$.



Convolutional encoder

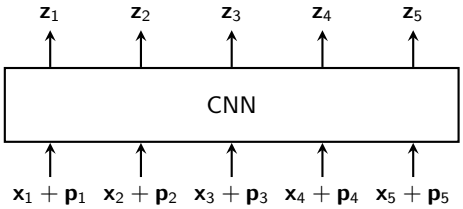
- Gehring et al. (2017) proposed to use a convolutional network (CNN) to encode input sequences.
- Since convolutional layers have shared weights, they can process sequences of varying lengths.



- Advantage: CNN can compute representations in all positions in parallel. We use both preceding and subsequent positions (unlike a bi-directional RNN).
- Disadvantage: CNN does not take into account whether the position is at the beginning of a sequence or at the end.

Convolutional encoder

- Gehring et al. (2017) proposed to use a convolutional network (CNN) to encode input sequences.
- Since convolutional layers have shared weights, they can process sequences of varying lengths.

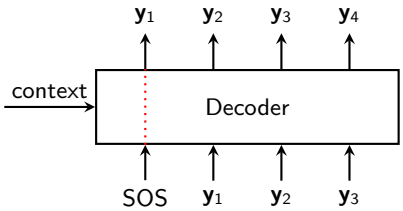


- Advantage: CNN can compute representations in all positions in parallel. We use both preceding and subsequent positions (unlike a bi-directional RNN).
- Disadvantage: CNN does not take into account whether the position is at the beginning of a sequence or at the end.
- Gehring et al. (2017) fix this problem by adding position embedding $\mathbf{p}_j$ to word embeddings $\mathbf{x}_j$.

Decoding with convolutional layers

- Can we also avoid using RNNs in the decoder?
- The decoder is an autoregressive model with the context provided by the encoder

$$\mathbf{y}_i = f(\mathbf{y}_{i-1}, ..., \mathbf{y}_1, \mathbf{z}_1, ..., \mathbf{z}_n)$$

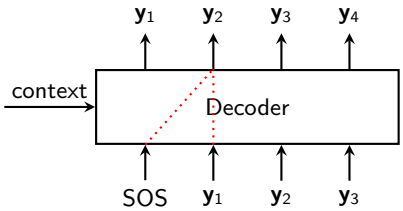


- This function can be modeled by a (convolutional) network:
  - Inputs and outputs are same sequences but 1) the output is shifted by one position, 2) the input sequence starts with a special SOS token.
  - The receptive field of $\mathbf{y}_i$ should not contain subsequent elements $\mathbf{y}_{i'}$, $i' \geq i$ (this can be achieved by using shifted convolutions).
  - Since we process sequences (inputs with one-dimensional structure), we use 1d convolutions.

Decoding with convolutional layers

- Can we also avoid using RNNs in the decoder?
- The decoder is an autoregressive model with the context provided by the encoder

$$\mathbf{y}_i = f(\mathbf{y}_{i-1}, ..., \mathbf{y}_1, \mathbf{z}_1, ..., \mathbf{z}_n)$$

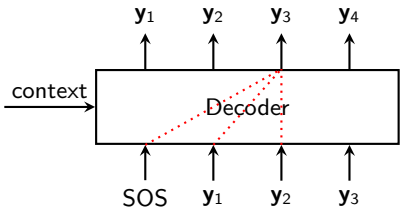


- This function can be modeled by a (convolutional) network:
  - Inputs and outputs are same sequences but 1) the output is shifted by one position, 2) the input sequence starts with a special SOS token.
  - The receptive field of $\mathbf{y}_i$ should not contain subsequent elements $\mathbf{y}_{i'}$, $i' \geq i$ (this can be achieved by using shifted convolutions).
  - Since we process sequences (inputs with one-dimensional structure), we use 1d convolutions.

Decoding with convolutional layers

- Can we also avoid using RNNs in the decoder?
- The decoder is an autoregressive model with the context provided by the encoder

$$\mathbf{y}_i = f(\mathbf{y}_{i-1}, ..., \mathbf{y}_1, \mathbf{z}_1, ..., \mathbf{z}_n)$$

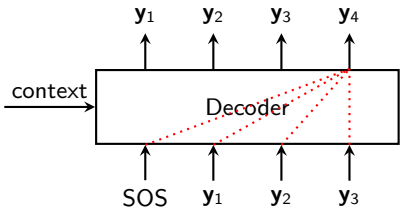


- This function can be modeled by a (convolutional) network:
  - Inputs and outputs are same sequences but 1) the output is shifted by one position, 2) the input sequence starts with a special SOS token.
  - The receptive field of $\mathbf{y}_i$ should not contain subsequent elements $\mathbf{y}_{i'}$, $i' \geq i$ (this can be achieved by using shifted convolutions).
  - Since we process sequences (inputs with one-dimensional structure), we use 1d convolutions.

Decoding with convolutional layers

- Can we also avoid using RNNs in the decoder?
- The decoder is an autoregressive model with the context provided by the encoder

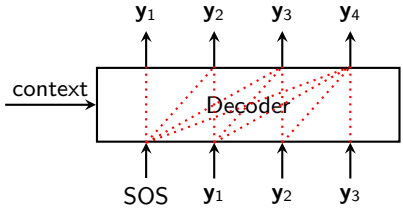
$$\mathbf{y}_i = f(\mathbf{y}_{i-1}, ..., \mathbf{y}_1, \mathbf{z}_1, ..., \mathbf{z}_n)$$



- This function can be modeled by a (convolutional) network:
  - Inputs and outputs are same sequences but 1) the output is shifted by one position, 2) the input sequence starts with a special SOS token.
  - The receptive field of $\mathbf{y}_i$ should not contain subsequent elements $\mathbf{y}_{i'}$, $i' \geq i$ (this can be achieved by using shifted convolutions).
  - Since we process sequences (inputs with one-dimensional structure), we use 1d convolutions.

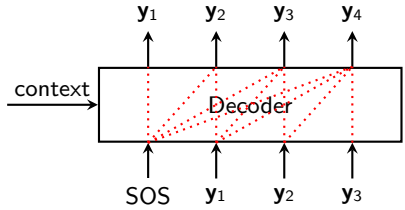
Decoding with convolutional layers

- Advantage of a convolutional decoder: During training, we can compute output elements for all positions *in parallel*. Recall that in the RNN decoder, we had to produce the output sequence one element at a time.

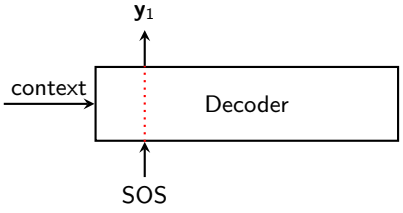


Decoding with convolutional layers

- Advantage of a convolutional decoder: During training, we can compute output elements for all positions *in parallel*. Recall that in the RNN decoder, we had to produce the output sequence one element at a time.

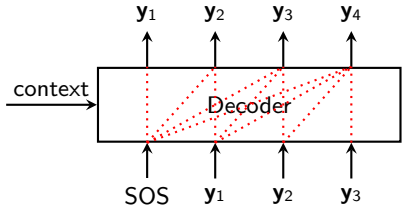


- At test time (generation mode), we still have to produce the output sequence one element at a time (since it is an autoregressive model).

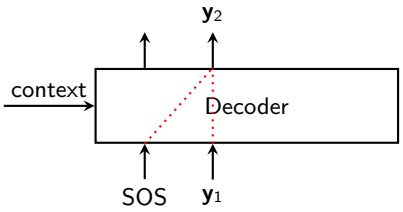


Decoding with convolutional layers

- Advantage of a convolutional decoder: During training, we can compute output elements for all positions *in parallel*. Recall that in the RNN decoder, we had to produce the output sequence one element at a time.



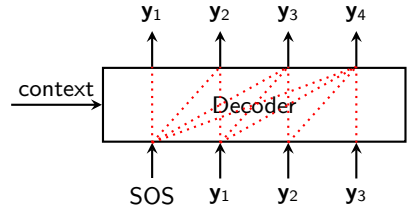
- At test time (generation mode), we still have to produce the output sequence one element at a time (since it is an autoregressive model).

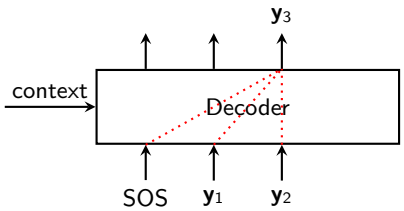


Decoding with convolutional layers

- Advantage of a convolutional decoder: During training, we can compute output elements for all positions *in parallel*. Recall that in the RNN decoder, we had to produce the output sequence one element at a time.

- At test time (generation mode), we still have to produce the output sequence one element at a time (since it is an autoregressive model).

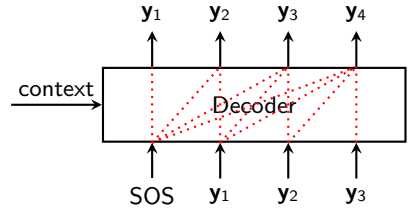


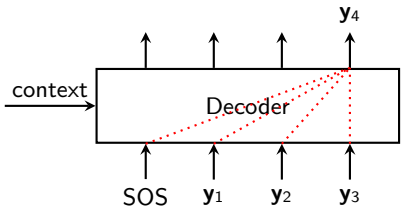


Decoding with convolutional layers

- Advantage of a convolutional decoder: During training, we can compute output elements for all positions *in parallel*. Recall that in the RNN decoder, we had to produce the output sequence one element at a time.

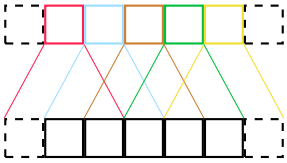
- At test time (generation mode), we still have to produce the output sequence one element at a time (since it is an autoregressive model).



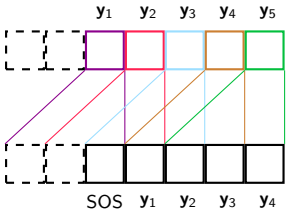


An autoregressive model with 1d convolutional layer

- We can make sure that the receptive field of $\mathbf{y}_i$ does not contain subsequent elements $\mathbf{y}_{i'}$, $i' \geq i$ by using shifted convolutions.



standard convolution



shifted convolution

- If we stack multiple convolutional layers built in the same way, the desired property is preserved.

Attention in a convolutional decoder

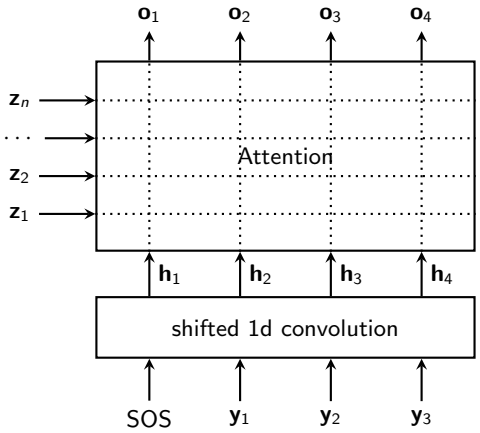
- How can we use the context provided by the encoder in such a decoder?
- Attention used in (Gehring et al., 2017):

$$\mathbf{o}_i = \sum_{j=1} \alpha_{ij}(\mathbf{z}_j + \mathbf{x}_j + \mathbf{p}_j)$$

where $\mathbf{x}_j$ are word embeddings and $\mathbf{p}_j$ are position embeddings for the input sequence.

- The attention weights are

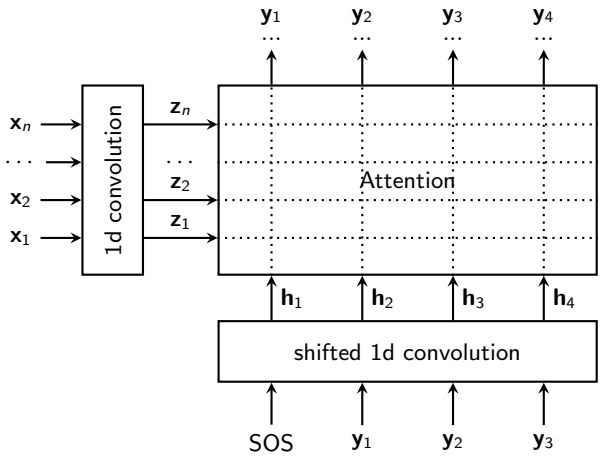
$$\alpha_{ij} = \frac{\exp(\mathbf{h}_i^\top \mathbf{z}_j)}{\sum_{j'=1}^{n} \exp(\mathbf{h}_i^\top \mathbf{z}_{j'})}$$



- Attention compares $\mathbf{h}_i$ to representations $\mathbf{z}_j$ using dot product and passes value $\mathbf{z}_j + \mathbf{e}_j$ corresponding to the best match.

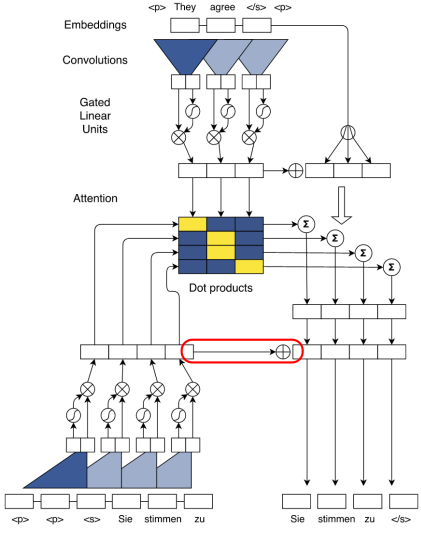
Full architecture from (Gehring et al., 2017)

- They used multiple decoder blocks stacked on top of each other.
- Each decoder block attends to the outputs of the encoder $\mathbf{z}_j$.
- There are skip connections (skipping the attention block).



Full architecture (Gehring et al., 2017)

- They used multiple decoder blocks stacked on top of each other.
- Each decoder block attends to the outputs of the encoder $\mathbf{z}_j$.
- There are skip connections (skipping the attention block).



Translation performance

The translation performances on an English-to-French translation task (WMT'14)

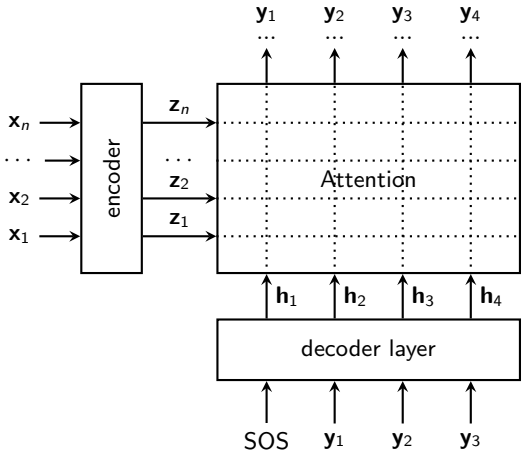
Model	BLEU
Simple Enc-Dec	17.82
Attention-based Enc-Dec	28.45
Attention-based Enc-Dec (LV)	34.11
Attention-based Enc-Dec (LV, ensemble)	37.19
ConvS2S (BPE 40K)	**40.51**

Transformers

(Vaswani et al., 2017)

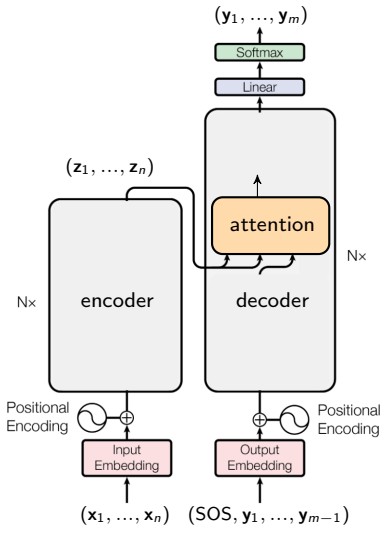
Transformer architecture

- The general architecture is similar to ConvS2S:
  - The encoder converts input sequence $(\mathbf{x}_1, ..., \mathbf{x}_n)$ into continuous representations $(\mathbf{z}_1, ..., \mathbf{z}_n)$.
  - The decoder processes all positions in parallel using shifted output sequence $(\mathbf{y}_1, ..., \mathbf{y}_m)$ as input and output. The autoregressive structure is preserved by masking.
  - The decoder attends to representations $(\mathbf{z}_1, ..., \mathbf{z}_n)$.



Transformer architecture

- The general architecture is similar to ConvS2S:
  - The encoder converts input sequence $(\mathbf{x}_1, ..., \mathbf{x}_n)$ into continuous representations $(\mathbf{z}_1, ..., \mathbf{z}_n)$.
  - The decoder processes all positions in parallel using shifted output sequence $(\mathbf{y}_1, ..., \mathbf{y}_m)$ as input and output. The autoregressive structure is preserved by masking.
  - The decoder attends to representations $(\mathbf{z}_1, ..., \mathbf{z}_n)$.



- Intuition from ConvS2S: The attention mechanism compares intermediate representations $\mathbf{h}_i$ developed in the decoder with encoded inputs $\mathbf{z}_j$ and outputs $\mathbf{z}_j$ that is closest to $\mathbf{h}_i$.

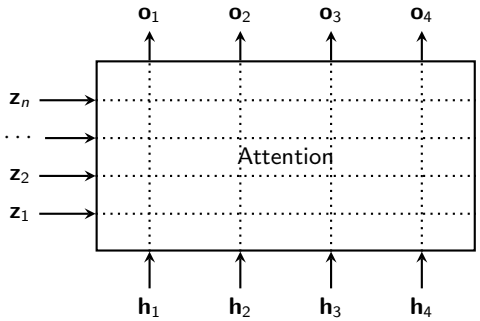
- Basic attention mechanism in transformers:

$$\mathbf{o}_i = \sum_{j=1}^{n} \alpha_{ij} \mathbf{z}_j$$

$$\alpha_{ij} = \frac{\exp(\mathbf{z}_j^\top \mathbf{h}_i / \sqrt{d_k})}{\sum_{j'=1}^{n} \exp(\mathbf{z}_{j'}^\top \mathbf{h}_i / \sqrt{d_k})}$$

  d_k is the dimensionality of the $\mathbf{z}_j$ and $\mathbf{h}_i$.

- The authors called this *scaled dot-product attention*.

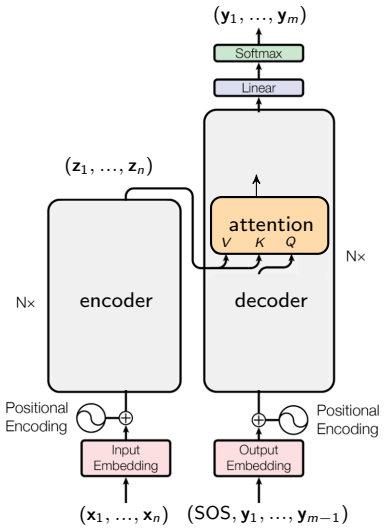


Transformer: Scaled dot-product attention

- We can think of the scaled dot-product attention as finding values $\mathbf{v}_j = \mathbf{z}_j$ with keys $\mathbf{k}_j = \mathbf{z}_j$ that are closest to query $\mathbf{q}_i = \mathbf{h}_i$.
- Re-writing the scaled dot-product attention using keys, values and query:

$$\mathbf{o}_i = \sum_{j=1}^{n} \alpha_{ij} \mathbf{v}_j$$

$$\alpha_{ij} = \frac{\exp(\mathbf{k}_j^\top \mathbf{q}_i / \sqrt{d_k})}{\sum_{j'=1}^{n} \exp(\mathbf{k}_{j'}^\top \mathbf{q}_i / \sqrt{d_k})}$$



Scaled dot-product attention

- Scaled dot-product attention:

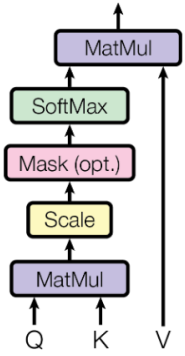
$$\mathbf{o}_i = \sum_{j=1}^{n} \alpha_{ij} \mathbf{v}_j$$

$$\alpha_{ij} = \frac{\exp(\mathbf{k}_j^\top \mathbf{q}_i / \sqrt{d_k})}{\sum_{j'=1}^{n} \exp(\mathbf{k}_{j'}^\top \mathbf{q}_i / \sqrt{d_k})}$$

in the matrix form:

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

with $\mathbf{V} \in \mathbb{R}^{m \times d_v}$, $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$, $\mathbf{K} \in \mathbb{R}^{m \times d_k}$.



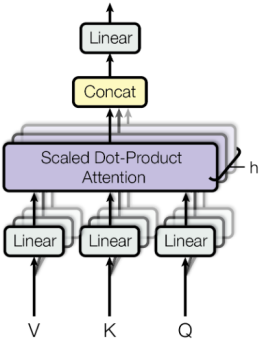
scaled dot-product attention

Multi-head attention

- Instead of doing a single scaled dot-product attention, the authors found it beneficial to project keys, queries and values into lower-dimensional spaces, perform scaled dot-product attention there and concatenate the outputs:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{head}_i = \text{attention}(QW_i^Q, KW_i^K, VW_i^V)$$

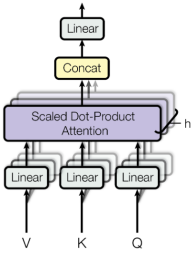
$\mathbf{V} \in \mathbb{R}^{m \times d_v}$, $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$, $\mathbf{K} \in \mathbb{R}^{m \times d_k}$,
$\text{head}_i \in \mathbb{R}^{n \times d_i}$, output $\in \mathbb{R}^{n \times d_k}$.

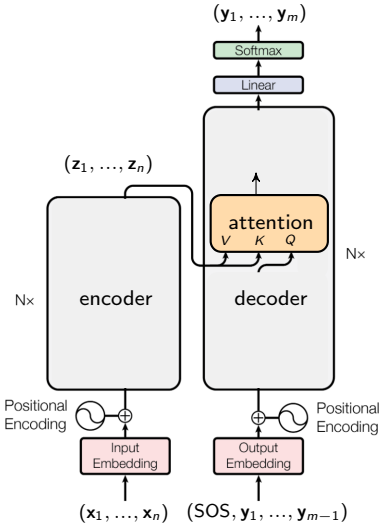


multi-head attention

Encoder-decoder attention

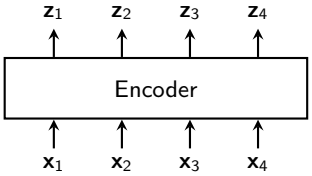
- Multi-head attention is used to attend to the encoder outputs by the decoder.





Transformer encoder: Self-attention

- How to implement the encoder? Previously we used: 1) bi-directional RNN, 2) convolutional network.
- Attention is all you need: Use the same multi-head attention mechanism to convert inputs $(\mathbf{x}_1, ..., \mathbf{x}_n)$ into representations $(\mathbf{z}_1, ..., \mathbf{z}_n)$.



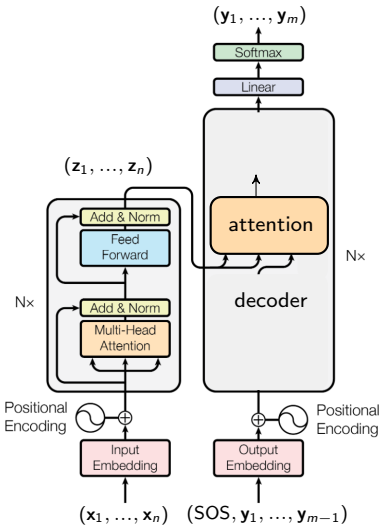
- For simplicity, assume that we use scaled dot-product attention:

$$\mathbf{z}_i = \sum_{j=1}^{n} \alpha_{ij} \mathbf{x}_j \qquad \alpha_{ij} = \frac{\exp(\mathbf{x}_j^\top \mathbf{x}_i / \sqrt{d_k})}{\sum_{j'=1}^{n} \exp(\mathbf{x}_{j'}^\top \mathbf{x}_i / \sqrt{d_k})}$$

- Thus, we use vectors $\mathbf{x}_i$ as keys, values and queries. This is called *self-attention*.
- Advantage: The first position affects the representation in the last position (and vice versa) already after one layer! (think how many layers are needed for that in RNN or convolutional encoders).

Transformer encoder

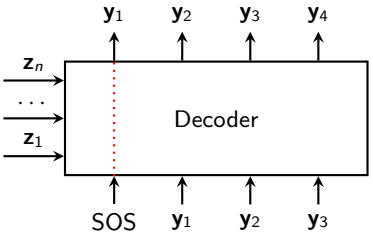
- After self-attention, the representation in each position is processed with a mini-MLP (Feed Forward block in the figure).
- The encoder is a stack of multiple such blocks (each block contains an attention module and a mini-MLP).
- Each block contains standard deep learning tricks:
  - skip connections
  - layer normalization



Transformer decoder

- Similarly to ConvS2S, the decoder implements an autoregressive model with the context provided by the encoder

$$\mathbf{y}_i = f(\mathbf{y}_{i-1}, ..., \mathbf{y}_1, \mathbf{z}_1, ..., \mathbf{z}_n)$$

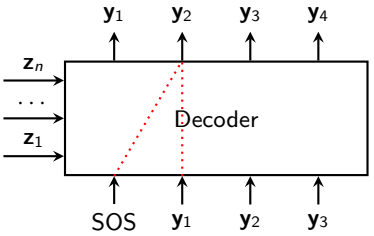


- When predicting word $\mathbf{y}_i$ we can use the preceding words $\mathbf{y}_1, ..., \mathbf{y}_{i-1}$ but not subsequent words $\mathbf{y}_i, ..., \mathbf{y}_m$.

Transformer decoder

- Similarly to ConvS2S, the decoder implements an autoregressive model with the context provided by the encoder

$$\mathbf{y}_i = f(\mathbf{y}_{i-1}, ..., \mathbf{y}_1, \mathbf{z}_1, ..., \mathbf{z}_n)$$

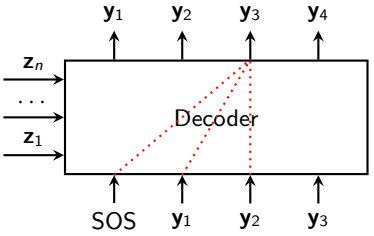


- When predicting word $\mathbf{y}_i$ we can use the preceding words $\mathbf{y}_1, ..., \mathbf{y}_{i-1}$ but not subsequent words $\mathbf{y}_i, ..., \mathbf{y}_m$.

Transformer decoder

- Similarly to ConvS2S, the decoder implements an autoregressive model with the context provided by the encoder

$$\mathbf{y}_i = f(\mathbf{y}_{i-1}, ..., \mathbf{y}_1, \mathbf{z}_1, ..., \mathbf{z}_n)$$

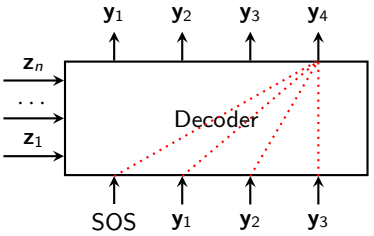


- When predicting word $\mathbf{y}_i$ we can use the preceding words $\mathbf{y}_1, ..., \mathbf{y}_{i-1}$ but not subsequent words $\mathbf{y}_i, ..., \mathbf{y}_m$.

Transformer decoder

- Similarly to ConvS2S, the decoder implements an autoregressive model with the context provided by the encoder

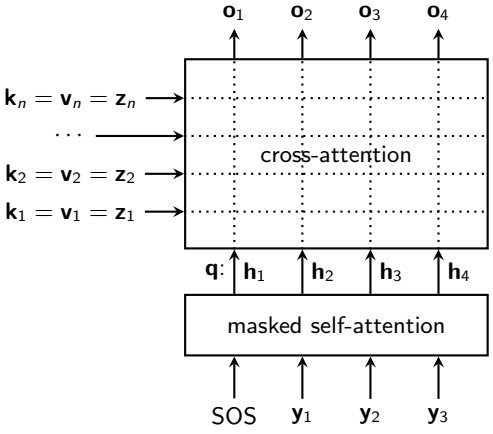
$$\mathbf{y}_i = f(\mathbf{y}_{i-1}, ..., \mathbf{y}_1, \mathbf{z}_1, ..., \mathbf{z}_n)$$



- When predicting word $\mathbf{y}_i$ we can use the preceding words $\mathbf{y}_1, ..., \mathbf{y}_{i-1}$ but not subsequent words $\mathbf{y}_i, ..., \mathbf{y}_m$.

Transformer decoder

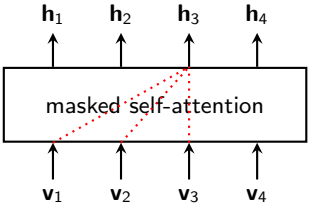
- The cross-attention block is the multi-head attention module described earlier.
- Again, attention-is-all-you-need idea: Use self-attention as a building block of the decoder.
- We need to make sure that we do not use subsequent inputs $\mathbf{y}_i, ..., \mathbf{y}_m$ when producing output $\mathbf{o}_i$ at position i. This is done using masked self-attention (see next slide).



Transformer decoder: Masked self-attention

- Let us denote the inputs of the self-attention layer as $\mathbf{v}_j$ and outputs as $\mathbf{h}_j$.
- For simplicity, assume that we use scaled dot-product attention:

$$\mathbf{h}_i = \sum_{j=1}^{m} \alpha_{ij} \mathbf{v}_j \qquad \alpha_{ij} = \frac{\exp(\mathbf{v}_j^\top \mathbf{v}_i / \sqrt{d_k} + m_{ij})}{\sum_{j'=1}^{m} \exp(\mathbf{v}_{j'}^\top \mathbf{v}_i / \sqrt{d_k} + m_{ij'})}$$

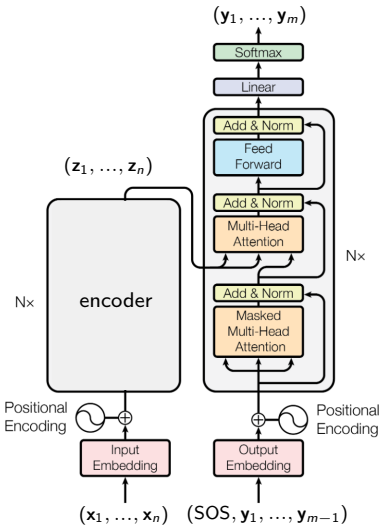


- We want not to use subsequent positions $\mathbf{v}_{i+1}, ..., \mathbf{v}_m$ when computing output $\mathbf{h}_i$. We can do that using attention masks m_{ij}:

$$m_{ij} = 0\,, \text{ if } j \leq i$$
$$m_{ij} = -\infty \text{ and therefore } \alpha_{ij} = 0, \text{ if } j > i$$

Decoder

- After self-attention and cross-attention, the representation in each position is processed with a mini-MLP (Feed Forward block in the figure).
- The decoder is a stack of multiple such blocks (each block contains two attention modules and a mini-MLP).
- Each block contains standard deep learning tricks:
  - skip connections
  - layer normalization

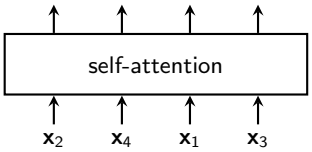


Transformer's positional encoding

- For simplicity, assume that we use scaled dot-product attention:

$$\mathbf{z}_i = \sum_{j=1}^{n} \alpha_{ij} \mathbf{x}_j \qquad \alpha_{ij} = \frac{\exp(\mathbf{x}_j^\top \mathbf{x}_i / \sqrt{d_k})}{\sum_{j'=1}^{n} \exp(\mathbf{x}_{j'}^\top \mathbf{x}_i / \sqrt{d_k})}$$

  What will happen to the outputs, if we shuffle the inputs (change their order)?

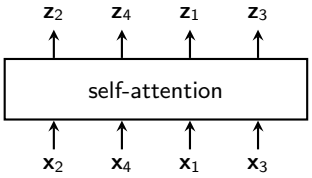


Transformer's positional encoding

- For simplicity, assume that we use scaled dot-product attention:

$$\mathbf{z}_i = \sum_{j=1}^{n} \alpha_{ij} \mathbf{x}_j \qquad \alpha_{ij} = \frac{\exp(\mathbf{x}_j^\top \mathbf{x}_i / \sqrt{d_k})}{\sum_{j'=1}^{n} \exp(\mathbf{x}_{j'}^\top \mathbf{x}_i / \sqrt{d_k})}$$

  What will happen to the outputs, if we shuffle the inputs (change their order)?



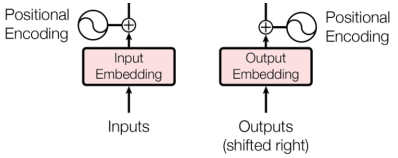
- The outputs will be shuffled in the same way. Thus, the computed representations will not depend on the order of the elements in the input sequence.
- This is not desired: the order of the words is important for understanding the meaning of a sentence.

Transformer's positional encoding

- Recall: ConvS2S used position embedding (embedding positions just like words) and added them to word embeddings.
- Transformers use hard-coded (not learned) positional encoding:

$$\mathrm{PE}(p, 2i) = \sin(p/10000^{2i/d})$$
$$\mathrm{PE}(p, 2i+1) = \cos(p/10000^{2i/d})$$

  where p is position, i is the element of the encoding.
- This encoding has the same dimensionality d as input/output embeddings.

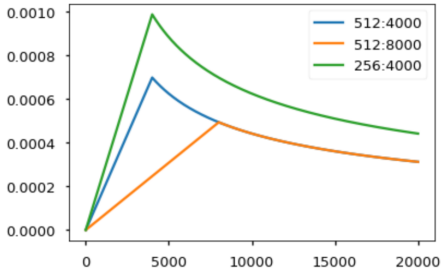


source: Annotated Transformer

- Motivation: It is easy for the model to learn to attend by relative positions, since for any fixed offset k, PE_{p+k} can be represented as a linear function of PE_p.

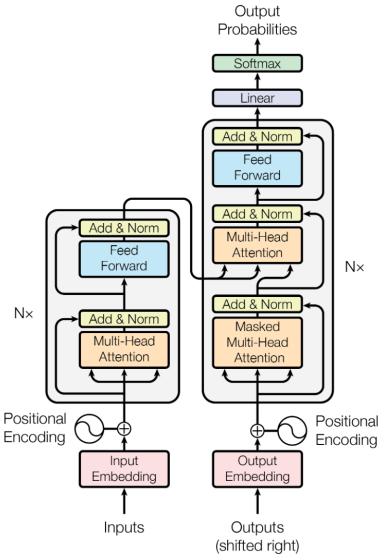
Transformer: Full model

- Training of the transformer model needs a ramp-up of the learning rate:



source: Annotated transformer

- If you have trouble understanding the model, check out the Annotated Transformer blog post.



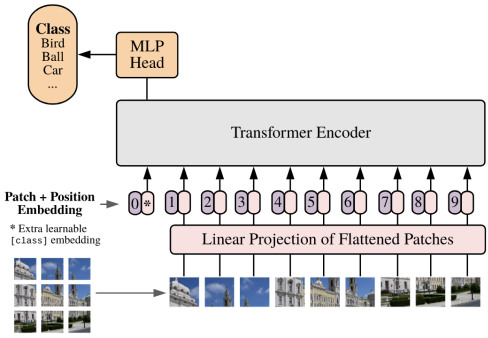
Translation performance

The translation performances on an English-to-French translation task (WMT'14) according to (Vaswani et al., 2017)

Model	BLEU
ConvS2S	40.46
ConvS2S (ensemble)	41.29
Transformer (base model)	38.1
Transformer (big)	**41.8**

Vision Transformer (Dosovitskiy et al., 2020)

- Transformers have been used in multiple domains showing great performance.
- Vision Transformer (ViT): A transformer applied to image classification tasks
  - split an image into fixed-size patches
  - linearly embed each of them
  - add position embeddings
  - feed the resulting sequence of vectors to a standard Transformer encoder.
  - In order to perform classification, add an extra learnable "classification token" to the sequence.



- ViT is pre-trained on large datasets and then fine-tuned to (smaller) downstream tasks.

BERT: Transformer-based language model

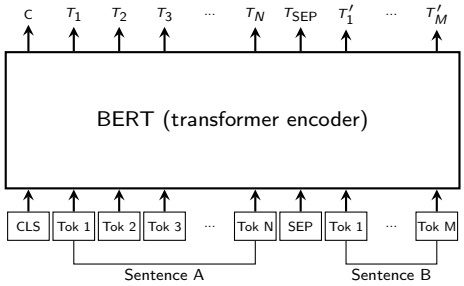
(Devlin et al., 2018)

Transfer learning in natural language processing

- Common natural language understanding tasks (see, e.g., GLUE benchmark):
  - Sentiment analysis: e.g., classification of sentences extracted from movie reviews
  - Question answering: e.g., detect pairs (question, sentence) which contain the correct answer
  - Determine if two sentences are semantically equivalent (binary classification)
- Labeled datasets for such tasks are often limited (labels are expensive to collect).
- Transfer learning:
  - Pre-train language models on large text corpora (unlabeled data, thus unsupervised learning)
  - Fine-tune a pre-trained model to a specific task

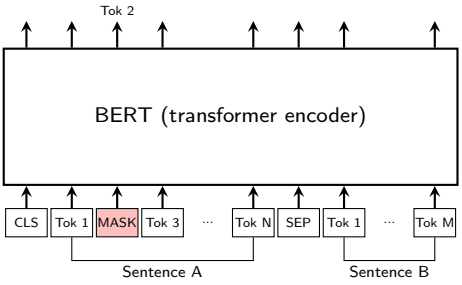
BERT: Pre-trained language model (Devlin et al., 2018)

- The model is essentially a transformer encoder (Vaswani et al., 2017).
- The model can represent either a single sentence or a pair of sentences (we need to process pairs in some downstream tasks, such as question answering).
- Pre-trained on a large corpus, e.g., English Wikipedia (2,500M words).



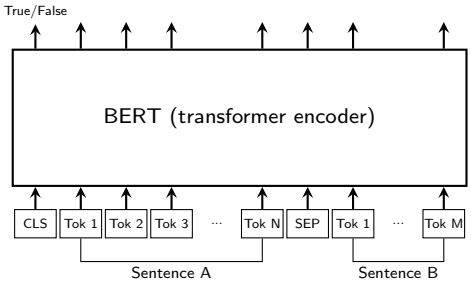
BERT: Pre-training task 1

- Pre-training task 1: Predict a masked input token (denoising task).
  - Use special MASK token to "corrupt" the input sequence.
  - The task is to reconstruct the masked token.



BERT: Pre-training task 2

- Pre-training task 2: Predict whether sentence B follows sentence A.
  - 50% of the time sentence B follows sentence A in the corpus.
  - 50% of the time sentence B is randomly chosen.
  - Binary classification task.

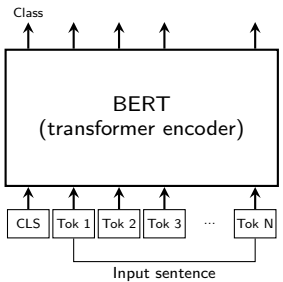


Fine-tuning BERT on a sentence classification task

Sentence: Although the value added services being provided are great but the prices are high. **Class:** mixed review

Sentence: Great work done #XYZ Problem resolved by customer care in just one day. **Class:** postive review

- BERT is fine-tuned on task-specific training data: task-specific inputs and outputs
- Example: sentence classification task:
  - Sentence A: input sentence
  - Sentence B: ∅
  - Output: target class (taken from the first position)
- New layer is introduced to convert the output at the first position into class probabilities.
- All parameters of the model are fine-tuned!



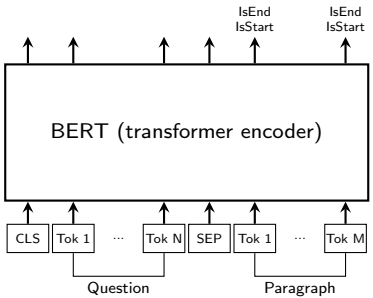
Fine-tuning BERT on a question answering task

Paragraph: Beyoncé Giselle Knowles-Carter (born September 4, 1981) is an American singer, songwriter, record producer, dancer and actress. Born and raised in Houston, Texas, Beyoncé performed in various singing and dancing competitions as a child. She rose to fame in the late 1990s as the lead singer of Destiny's Child, one of the best-selling girl groups of all time.

Question: When did Beyonce start becoming popular?

Correct answer: in the late 1990s

- Fine-tuning BERT on a question answering task:
  - Sentence A: question
  - Sentence B: paragraph (passage)
  - Output sequence: Probabilities of each word in the passage being the start and the end
- All parameters of the model are updated on the task-specific data!



BERT: GLUE Test results

GLUE Test results

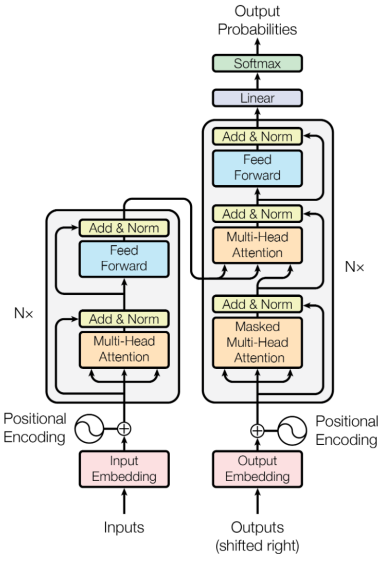
System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	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**

The number below each task denotes the number of training examples.
F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B,
and accuracy scores are reported for the other tasks.

Home assignment

Assignment 05_transformer

- You need to implement and train a transformer model for the task of statistical machine translation task (the same task as in the previous assignment).



Recommended reading

- Papers cited in the lecture slides.
- The Annotated Transformer blog post.