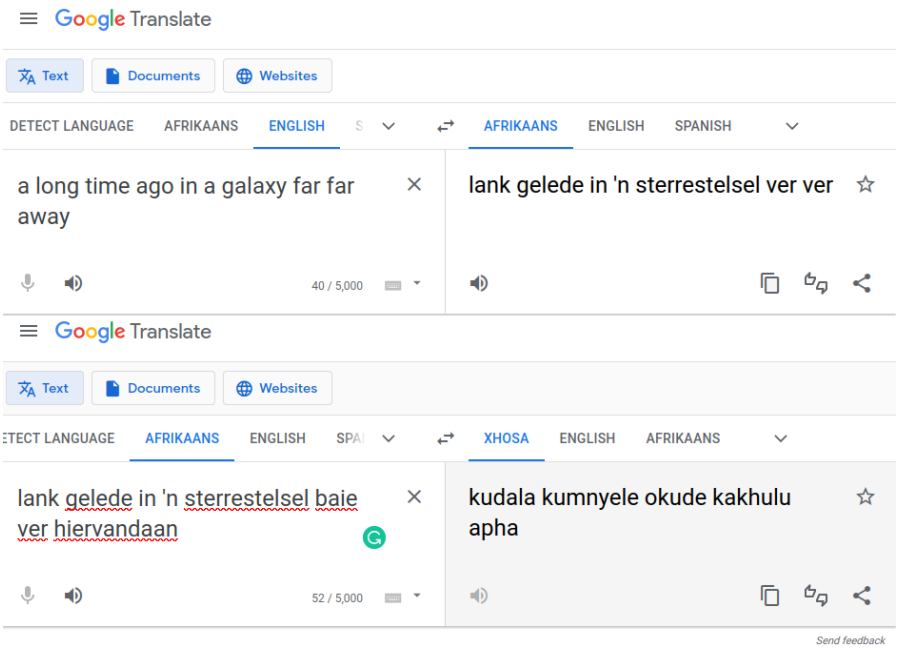
# Encoder-decoder models and attention

Herman Kamper

# Machine translation



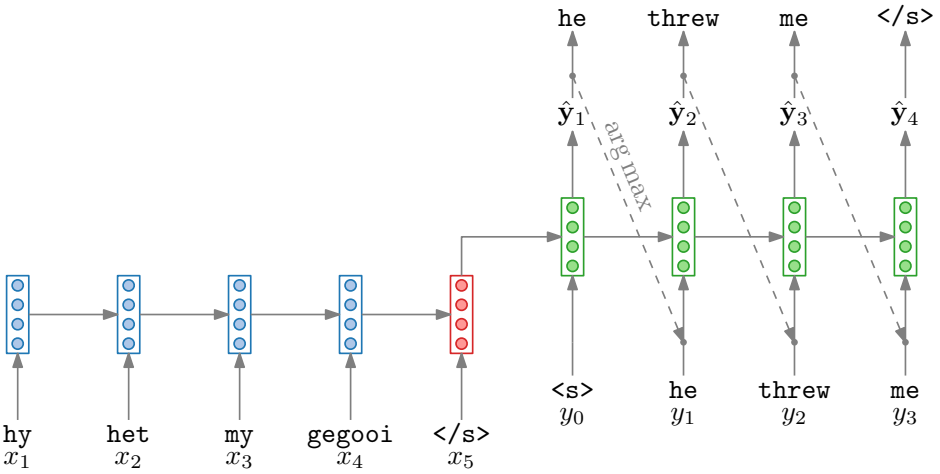

Old-school machine translation (MT) was done with big complex models with several subsystems in a paradigm referred to as statistical machine translation (SMT).

The most recent paradigm of MT models, starting in around 2014, is referred to as neural machine translation (NMT).

In this note, we will use the running example of NMT as a way to look at encoder-decoder models (also called sequence-to-sequence models) and attention.

# An encoder-decoder RNN for MT

NMT is often framed as a sequence-to-sequence learning problem. The particular architecture often used within this learning framework is called an encoder-decoder architecture.




Dashed line: Shows what happens at test time. The decoder output is used as the input to the next step.

# The (N)MT problem

We want to map input sentence $X = x_{1:N}$ in the source language to the output sentence $Y = y_{1:T}$ in the target language.

Convention: $y_0 =$ <s> and $x_N = y_T =$ </s>

The goal is to find

$$\arg\max_Y P_{\boldsymbol{\theta}}(Y|X)$$

We can decompose the conditional probability using the product rule:

$$P_{\boldsymbol{\theta}}(Y|X) =$$

$$= \prod_{t=1}^{T} P_{\boldsymbol{\theta}}(y_t|y_{1:t-1}, X)$$

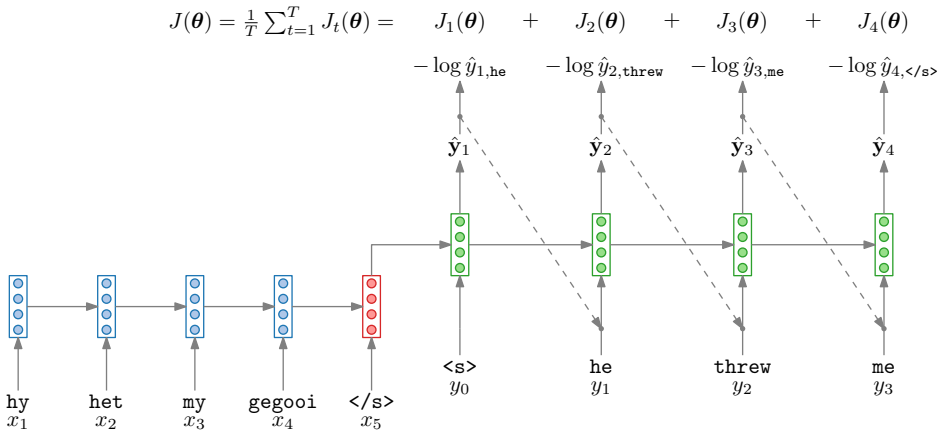
## Loss function

In NMT, we calculate $P_{\boldsymbol{\theta}}(Y|X)$ by outputting the conditional probability at every time step $t$:

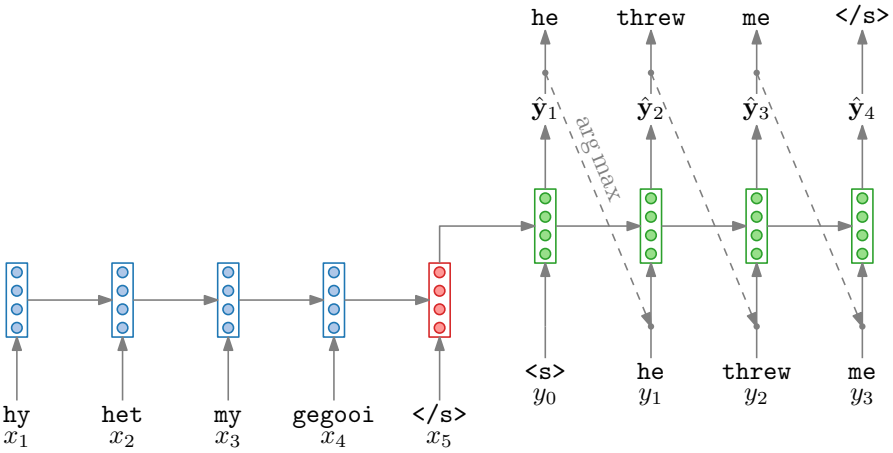
$$\hat{y}_{t,k} = P_{\boldsymbol{\theta}}(y_t = k|y_{1:t-1}, X)$$

We train the encoder-decoder model by optimizing the per-word negative log likelihood:

$$\begin{aligned} J(\boldsymbol{\theta}) &= -\frac{1}{T}\sum_{t=1}^{T} \log P_{\boldsymbol{\theta}}(y_t|y_{1:t-1}, X) \\ &= -\frac{1}{T}\sum_{t=1}^{T} \log \hat{y}_{t,y_t} = -\frac{1}{T}\sum_{t=1}^{T} \log J_t(\boldsymbol{\theta}) \end{aligned}$$


$J(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T} J_t(\boldsymbol{\theta}) = J_1(\boldsymbol{\theta}) + J_2(\boldsymbol{\theta}) + J_3(\boldsymbol{\theta}) + J_4(\boldsymbol{\theta})$
$-\log \hat{y}_{1,\texttt{he}}$
$-\log \hat{y}_{2,\texttt{threw}}$
$-\log \hat{y}_{3,\texttt{me}}$
$-\log \hat{y}_{4,\texttt{</s>}}$
$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_3$
$\hat{\mathbf{y}}_4$
hy $x_1$
het $x_2$
my $x_3$
gegooi $x_4$
</s> $x_5$
<s> $y_0$
he $y_1$
threw $y_2$
me $y_3$

# Encoder-decoder modelling choices



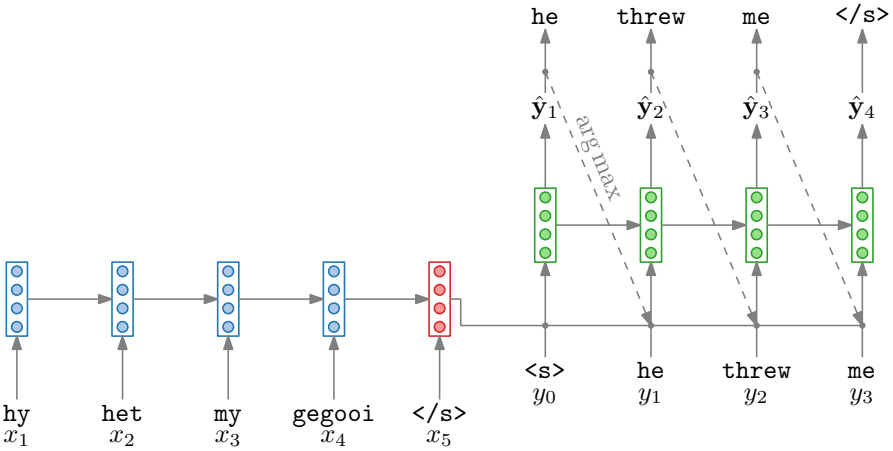

**Output conditioning at training and test time**

- Training time: We condition the decoder step $t$ on the ground truth word $y_{t-1}$ from the previous time step. This is called teacher forcing.
- Test time: We don't have the ground truth $y_{t-1}$. So we condition the decoder step $t$ on the predicted word $\hat{y}_{t-1}$ from the previous time step. We can take the $\arg\max$ as in the figures so far, or do something more fancy (later).

Training time: There are also training variants where you would sometimes use $\hat{y}_{t-1}$ during training to better match what happens at test time.

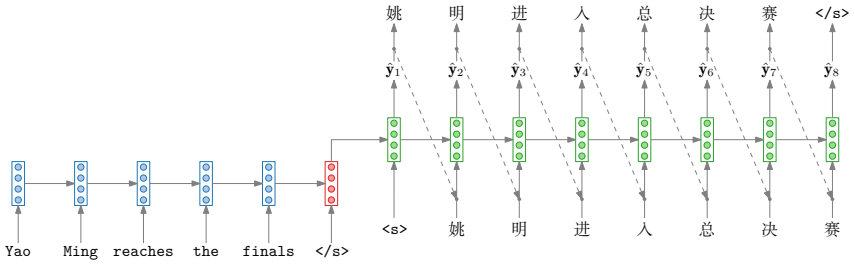
## Conditioning the decoder on the encoder output

- Above we used the last hidden vector from the encoder to initialize the decoder RNN.
- We could feed the final hidden representation from the encoder into some fully connected layers before conditioning the decoder.
- We could condition every decoder step on the encoder output:




## Output units: Words, characters, subwords

In the above we used words as output units. But we could also use characters: Maybe a good choice if the target language uses a non-Latin script. Or we could use subword units like BPE. This could address problems where some words are not in the vocabulary.



# Encoder-decoder: More general

### Encoder

Using $f$ to denote the transformation in the encoder's RNN, we can write the recurrance as:

$$\mathbf{h}_n = f(\mathbf{x}_n, \mathbf{h}_{n-1})$$

In the general case, the encoder transforms all its hidden states into a single fixed-dimensional representation:

$$\mathbf{c} = q(\mathbf{h}_{1:N})$$

In the examples above:

$$\mathbf{c} = \mathbf{h}_N$$

### Decoder

The decoder hidden state $\mathbf{s}_t$ depends on the previous model output, the previous decoder hidden state, and the encoder output:

$$\mathbf{s}_t = g(y_{t-1}, \mathbf{c}, \mathbf{s}_{t-1})$$

Normally the decoder hidden state $\mathbf{s}_t$ is passed on to some output operation in order to get

$$P_{\boldsymbol{\theta}}(y_t | y_{1:t-1}, \mathbf{c})$$

# Evaluating MT: BLEU

We want to compare a predicted translation $\hat{Y}$ to a reference $Y$.

Let $p_n$ denote the precision of $n$-grams of order $n$:

- Out of all the $n$-grams in $\hat{Y}$, how many of them occur in $Y$?
- Counts are capped by the number of occurrences in the reference. E.g. if $Y =$ `b b` and $\hat{Y} =$ `b b b`, then $p_1 = \frac{2}{3}$ since $\hat{Y}$ can only get credit for unigram `b` up to its count in $Y$ (which is two).

Example from Zhang et al. (2021):

$$Y = \texttt{the cat sat on the mat}$$
$$\hat{Y} = \texttt{the cat cat sat on}$$

Then $p_1 = \frac{4}{5}$, $p_2 = \frac{3}{4}$, $p_3 = \frac{1}{3}$ and $p_4 = \frac{0}{2}$.

Let $|Y|$ denote the sequence length, e.g. if $Y = y_{1:T}$ then $|Y| = T$.

The BLEU score is defined as (Papineni et al. 2002):

$$\boxed{\text{BLEU} = \exp\left\{\min\left(0, 1 - \frac{|Y|}{|\hat{Y}|}\right)\right\} \prod_{n=1}^{N} p_n^{1/2^n}}$$

where $N$ is the longest $n$-gram used for matching.

- If the reference and prediction match, the BLEU is 1.
- Longer $n$-grams are more difficult, so assign these a larger weight: For a fixed $p_n$ we have $p_n^{1/2^n}$ increasing for larger $n$.
- Very short sequence tend to get high $p_n$, which is unwanted: Penalize these with the exponential term. E.g. when $N = 2$ with $Y =$ `a b c d e f` and $\hat{Y} =$ `a b`, although $p_1 = p_2 = 1$, the penality factor $\exp\left\{1 - \frac{6}{2}\right\} = 0.14$ lowers the BLEU.

In practice there is often more than one reference.

# Greedy decoding

At test time, we can translate some input $X$ by taking the $\arg\max$ at every step of the decoder:

$$\hat{y}_t = \arg\max_{w \in \mathcal{V}} P_{\boldsymbol{\theta}}(y_t = w | y_{1:t-1}, X)$$

But this might be short sighted!

Input:

```
hy het my met 'n tert geslaan
```

(`he hit me with a pie`)

Decoding:

- `he ...`
- `he hit ...`
- `he hit a ...`

At the last third decoding step, `a` is the most probable next word, given the previously generated outputs. But it might have been better to take a less-probable word at this third step, maybe getting higher overall probabilities at some later decoding step. But now we've selected `a` and there is no way to recover.

# Beam search

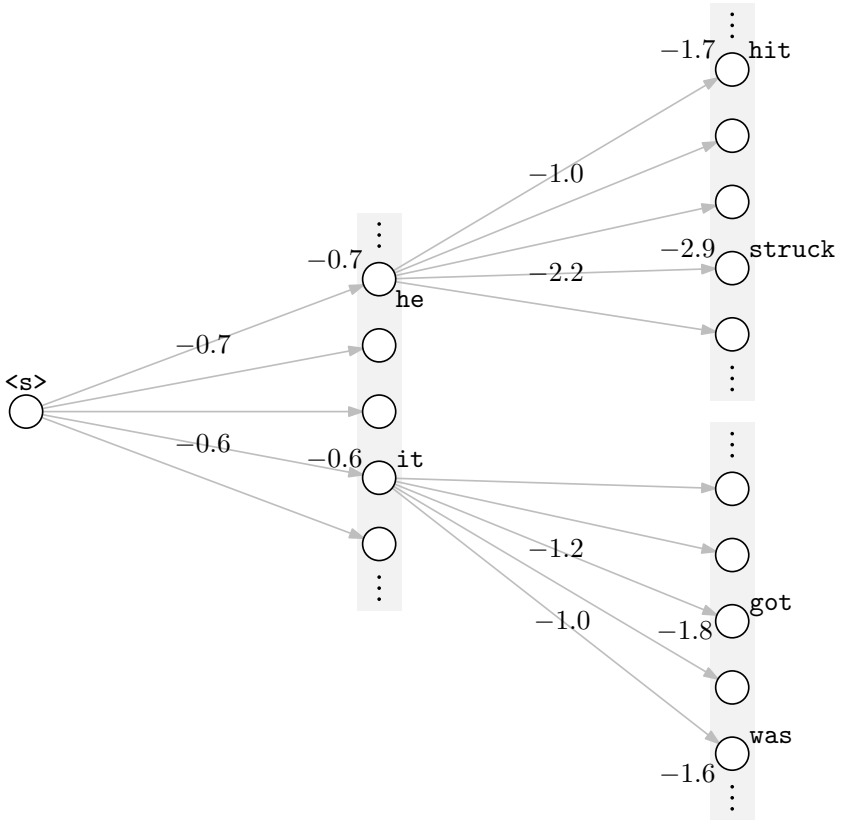
Beam search: On each decoding step, keep track of the $K$ most probable partial translations.

- $K$ is called the beam size ($5$ to $10$ for MT).
- The partial translation is called a hypothesis.

Beam search is not guaranteed to find overall optimal solution:

- But way more efficient than brute-forcing all paths (exhaustive search).
- Is likely to find a better solution than the greedy approach (if there is one). With $K = 1$, beam search is equivalent to greedy decoding.

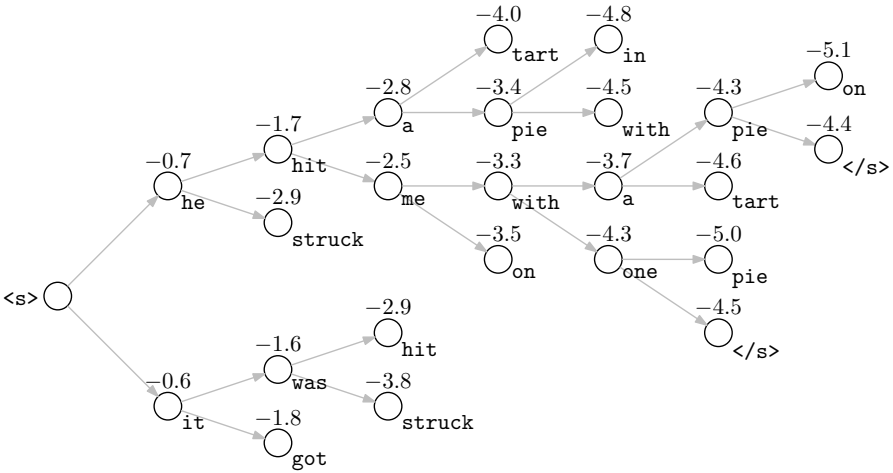
## Example: Beam search



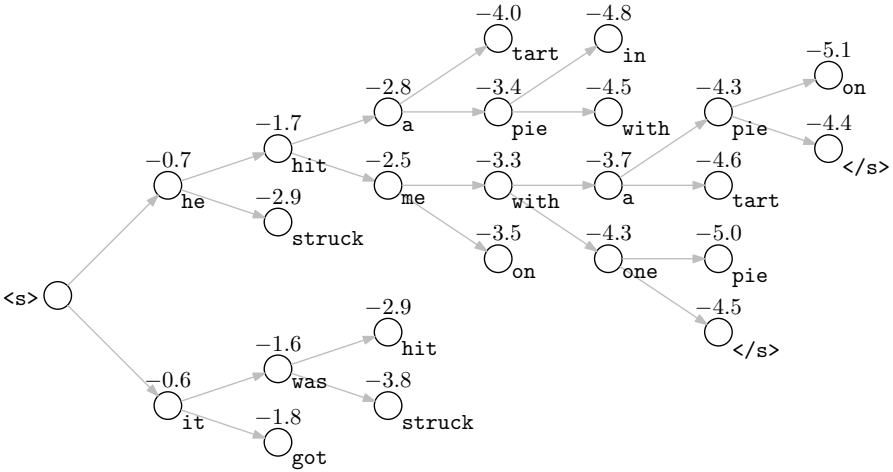

The score at each node is the log probability of that partial translation according to the model $\boldsymbol{\theta}$:

$$\begin{aligned} s(y_{1:t}) &= \log P_{\boldsymbol{\theta}}(y_{1:t}|X) \\ &= \log \sum_{i=1}^{t} P_{\boldsymbol{\theta}}(y_i|y_{1:i-1}, X) \end{aligned}$$

If we only show the top two nodes at every decoding step:



## What would have happened with greedy search?




Just looking at the first three decoding steps, we see that we would have continued with

`it was hit ...` $(-2.9)$

but would have missed

`he hit me ...` $(-2.5)$

because this better option only appeared later.

## Penalizing shorter hypotheses

Completed hypotheses could have different lengths, as in the example above if you consider the second most likely path. This can cause issues in some cases, since naive beam search would probably prefer shorter sequences for $\hat{Y}$ since this corresponds to adding together fewer log probability terms.

Some length normalization approach is therefore normally incorporated.

One simple approach: Normalize by the number of words in the hypothesis.

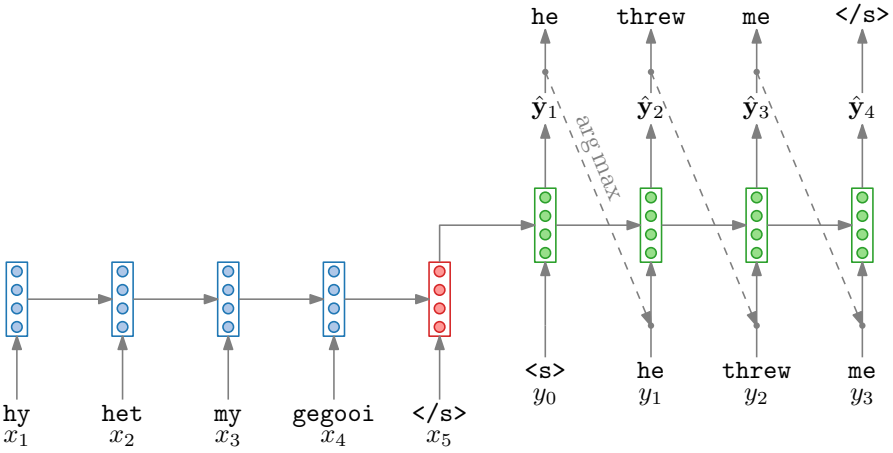
$$s(y_{1:t}) = \frac{1}{t} \log \sum_{i=1}^{t} P_{\boldsymbol{\theta}}(y_i | y_{1:i-1}, X)$$

# NMT with encoder-decoder summary

We have covered:

- Training
- Test-time decoding: Beam search
- Evaluation: BLEU

Is there anything left? Does this just always work? Any issues you can think of with the model? (Not just for NMT, but maybe for other problems as well?)



# Basic attention

(First skip this and look at the MT example on the next few pages. Then come back and map what you saw there to the equations here.)

We are at time step $t$ in the decoder.

- Encoder hidden states:

$$\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N \in \mathbb{R}^D$$

- Decoder hidden state at time step $t$:

$$\mathbf{s}_t \in \mathbb{R}^D$$

- Attention score for encoder hidden state $\mathbf{h}_n$:

$$a(\mathbf{s}_{t-1}, \mathbf{h}_n) = \mathbf{s}_{t-1}^\top \mathbf{h}_n \in \mathbb{R}$$

- Attention weight for encoder hidden state $\mathbf{h}_n$:

$$\begin{aligned}\alpha(\mathbf{s}_{t-1}, \mathbf{h}_n) &= \mathrm{softmax}_n\left(a(\mathbf{s}_{t-1}, \mathbf{h}_n)\right) \\ &= \frac{\exp\left\{a(\mathbf{s}_{t-1}, \mathbf{h}_n)\right\}}{\sum_{j=1}^{N} \exp\left\{a(\mathbf{s}_{t-1}, \mathbf{h}_j)\right\}} \in [0, 1]\end{aligned}$$

- Context vector at decoder time step $t$:

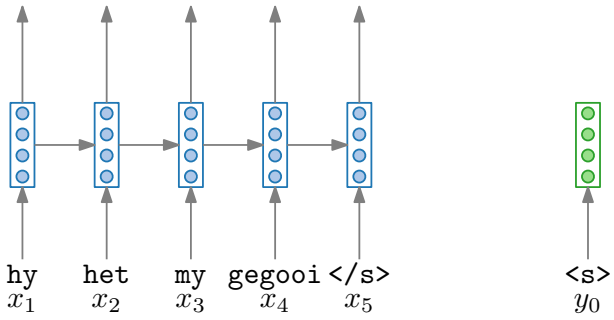
$$\mathbf{c}_t = \sum_{n=1}^{N} \alpha(\mathbf{s}_{t-1}, \mathbf{h}_n)\, \mathbf{h}_n \in \mathbb{R}^D$$

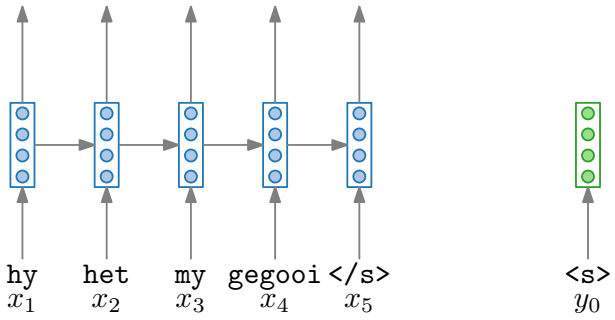
- Concatenate:

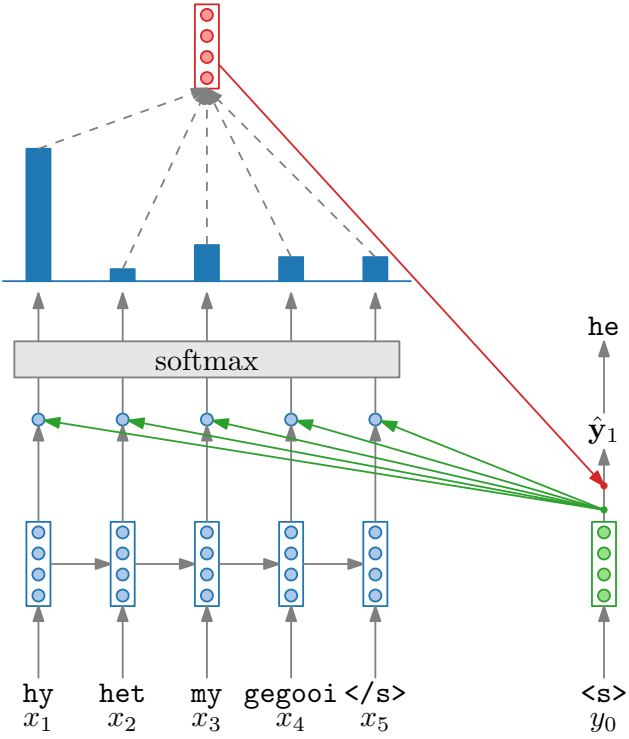
$$\Big[\mathbf{c}_t; \mathbf{s}_t\Big] \in \mathbb{R}^{2D}$$

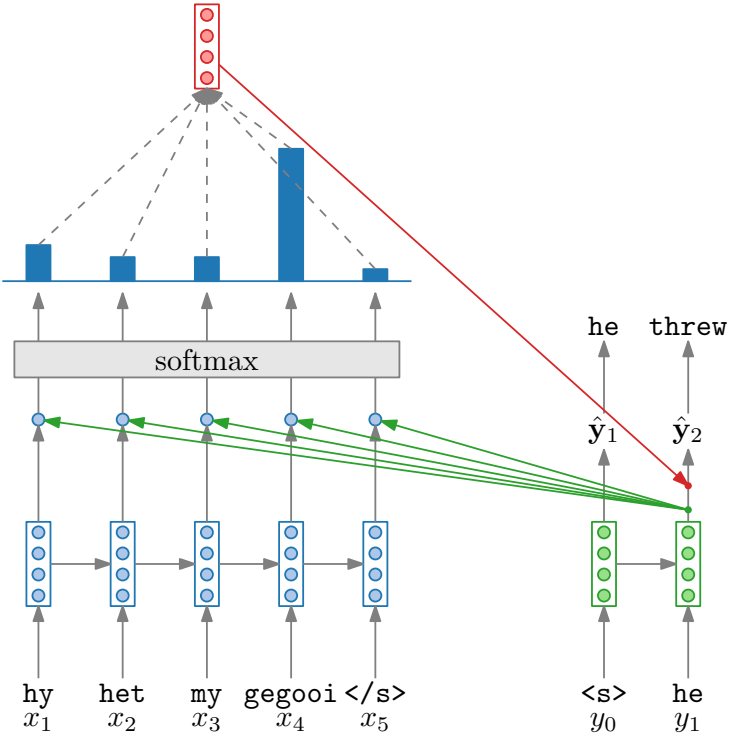
  and continue as in the non-attention decoding case, e.g.

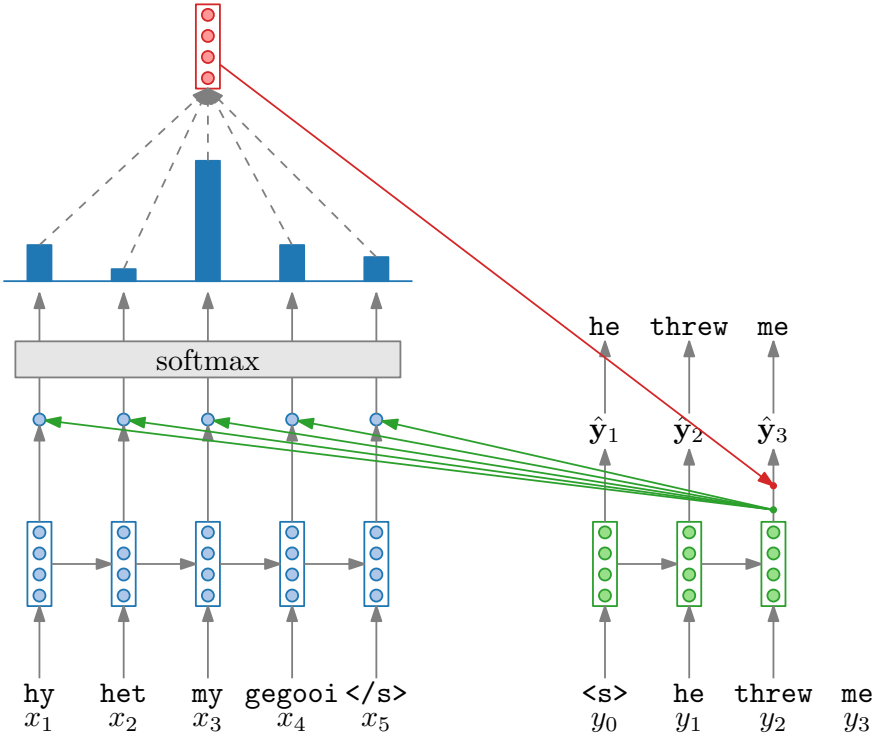
$$\hat{\mathbf{y}}_t = \mathrm{softmax}\left(\mathbf{W}_{ho}\Big[\mathbf{c}_t; \mathbf{s}_t\Big] + \mathbf{b}_o\right)$$


hy
$x_1$
het
$x_2$
my
$x_3$
gegooi
$x_4$
</s>
$x_5$
<s>
$y_0$


hy
$x_1$
het
$x_2$
my
$x_3$
gegooi
$x_4$
</s>
$x_5$
<s>
$y_0$

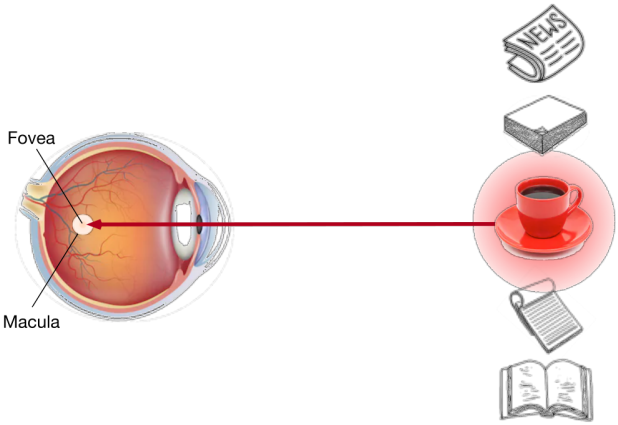

softmax
he
$\hat{\mathbf{y}}_1$
hy
$x_1$
het
$x_2$
my
$x_3$
gegooi
$x_4$
</s>
$x_5$
<s>
$y_0$


softmax
he
threw
$\hat{y}_1$
$\hat{y}_2$
hy
$x_1$
het
$x_2$
my
$x_3$
gegooi
$x_4$
</s>
$x_5$
<s>
$y_0$
he
$y_1$

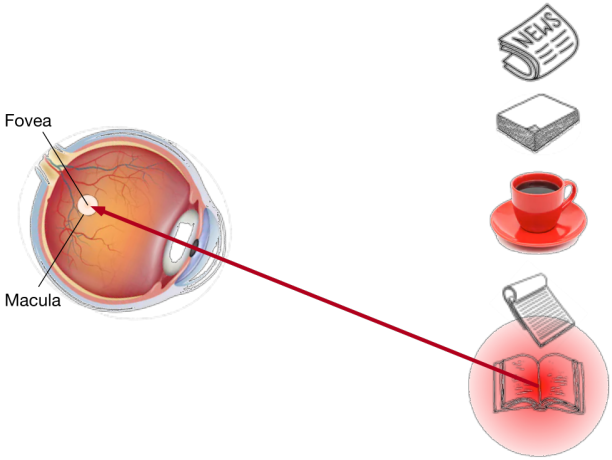

softmax
hy
$x_1$
het
$x_2$
my
$x_3$
gegooi
$x_4$
</s>
$x_5$
<s>
$y_0$
he
$y_1$
threw
$y_2$
me
$y_3$
he
threw
me
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$

# Attention: More general

Nonvolitional cues (keys):[1]




Volitional cues (queries):




[1]Figures from (Zhang et al. 2021).

## Attention with queries, keys and values

In general we have:

- Query $\mathbf{q} \in \mathbb{R}^{D_1}$: Volitional cues.
- Keys $\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_N \in \mathbb{R}^{D_2}$: Nonvolitional cues.
- Values $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N \in \mathbb{R}^{D}$: What is attended to.

The values $\mathbf{v}_n \in \mathbb{R}^D$ and the output context vector $\mathbf{c} \in \mathbb{R}^D$ have the same dimensionality. But the dimensionalities of the query $\mathbf{q} \in \mathbb{R}^{D_1}$ and keys $\mathbf{k}_n \in \mathbb{R}^{D_2}$ need not match, as long as there is a way to get the attention score $a$.

### Basic attention

In the basic version of attention that we looked at before, the values and the keys are the same: $\mathbf{v}_n = \mathbf{k}_n = \mathbf{h}_n$. I.e. the keys and values are the encoder hidden states (Bahdanau et al. 2014).

**All variants of attention have the following components**

- Output of attention: Context vector.

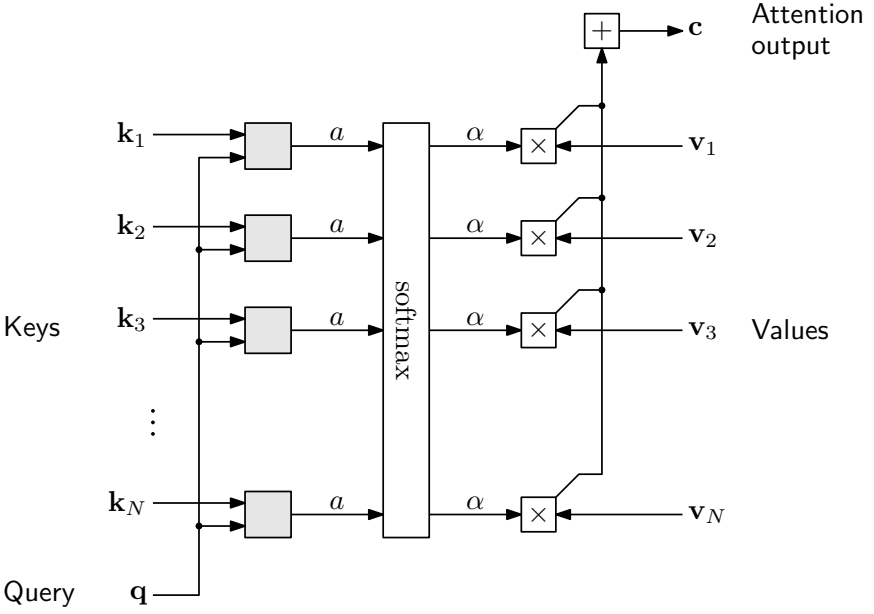
$$\mathbf{c} = \sum_{n=1}^{N} \alpha(\mathbf{q}, \mathbf{k}_n)\, \mathbf{v}_n \in \mathbb{R}^D$$

- Attention weight:

$$\begin{aligned} \alpha(\mathbf{q}, \mathbf{k}_n) &= \mathrm{softmax}_n\left(a(\mathbf{q}, \mathbf{k}_n)\right) \\ &= \frac{\exp\left\{a(\mathbf{q}, \mathbf{k}_n)\right\}}{\sum_{j=1}^{N} \exp\left\{a(\mathbf{q}, \mathbf{k}_j)\right\}} \in [0, 1] \end{aligned}$$

- Attention score:

$$a(\mathbf{q}, \mathbf{k}_n) \in \mathbb{R}$$



# Attention variants

## Different scoring options

Dimensionalities: Query $\mathbf{q} \in \mathbb{R}^{D_1}$ and keys $\mathbf{k} \in \mathbb{R}^{D_2}$.

- Dot product attention:

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\top \mathbf{k}$$

  Requires the dimensionalities $D_1 = D_2$ to match.

- Scaled dot product attention:

$$a(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q}^\top \mathbf{k}}{\sqrt{D_1}}$$

  Requires the dimensionalities $D_1 = D_2$ to match.

- Multiplicative attention:

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\top \mathbf{W} \mathbf{k}$$

  where $\mathbf{W} \in \mathbb{R}^{D_1 \times D_2}$

- Additive attention:

$$\begin{aligned} a(\mathbf{q}, \mathbf{k}) &= \mathrm{MLP}(\mathbf{q}, \mathbf{k}) \\ &= \mathbf{w}^\top \tanh\left(\mathbf{W}_q \mathbf{q} + \mathbf{W}_k \mathbf{k}\right) \end{aligned}$$

  with $\mathbf{w} \in \mathbb{R}^{D_3}$, $\mathbf{W}_q \in \mathbb{R}^{D_3 \times D_1}$ and $\mathbf{W}_k \in \mathbb{R}^{D_3 \times D_2}$.

## Different ways to use the context vector $\mathbf{c}_t$ in decoding

There are also different approaches to how and whether $\mathbf{c}_t$ gets passed on to the next hidden representation $\mathbf{h}_{t+1}$ in the decoder. See (Voita 2022) for details on two possible schemes.

# Transformers

- The annotated transformer
- D2L textbook
- CS224N: Transformers
- CS224N: Pretraining
- Lena Voita's blog

# Acknowledgements

This note relied very very heavily on content from:

- The CS224N course of Chris Manning at Stanford University.
- The D2L textbook from Zhang et al. (2021).

# References

D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ICLR*, 2015.

C. Manning, "CS224N: Machine translation, sequence-to-sequence and attention," *Stanford University*, 2022.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," *ACL*, 2002.

A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, 2021.