School of CIT
Social Computing Research Group

Technical
University
of Munich

Natural Language Processing
IN2361

Prof. Dr. Georg Groh

Chapter 10: Transformers and Pretrained Language Models

- content is based on [1] , [5]
- certain elements (e.g. equations or tables) were taken over or taken over in a modified form from [1] , [5]
- citations of [1] , [5] or from [1] , [5] are omitted for legibility
- errors are fully in the responsibility of Georg Groh
- BIG thanks to Dan and James for a great book!
- BIG thanks to Richard Socher, Chris Manning and their colleagues at Stanford for publishing materials [2],[5] of a great Deep NLP lecture

Self-Attention Networks

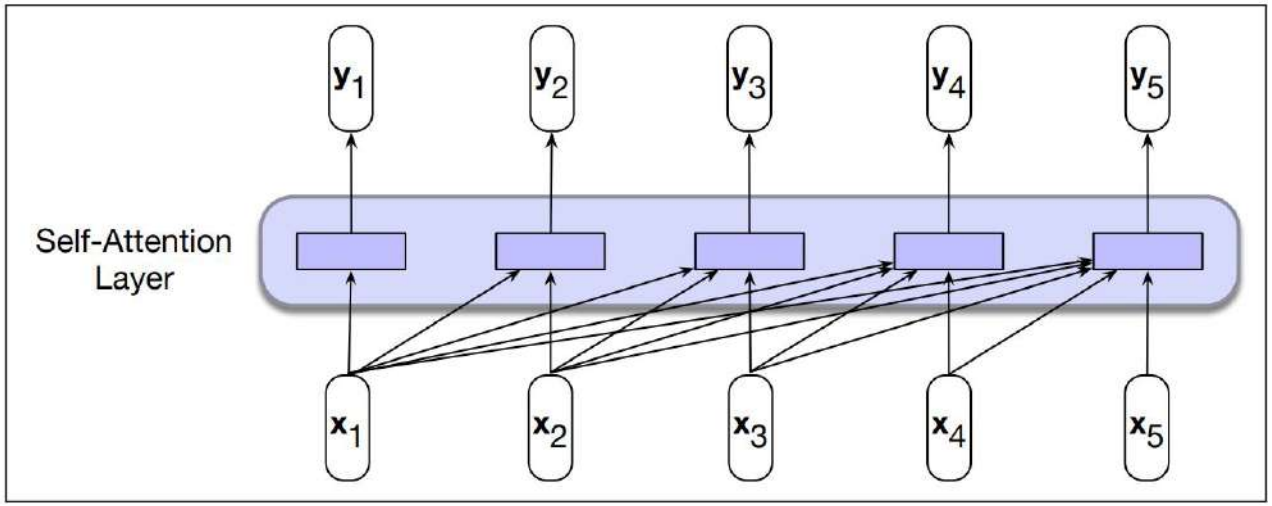
- Gates allow more distant information flow
- Core problem remains:
  - Series of recurrent connections leads to information loss + difficulties training
- Further computational problems:
  - Recurrent networks make parallelization hard

- Solution: Transformer networks
  - Mapping sequence of inputs to sequence of outputs

- Underlying mechanism: Self-attention

Self-Attention Networks

- Self-attention layer:
  - Extract information from arbitrary length context
  - No passing through intermediary recurrent connections



- For each input: Access to all information from previous inputs = availability of long term context
- Computation per node is independent = parallelizable

Self-Attention Networks

Basic attention mechanism: Scoring

- Comparing elements within a sequence in view of their relevance *for current computation*

  example: Computing y_3 = compare input x_3 to x_1 , x_2 and x_3

- Simple comparison: dot product

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

  result: Scalar from $-\infty$ to ∞

- Normalize scores into a weight vector a_{ij} using Softmax

$$\begin{aligned} \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \ \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \ \forall j \leq i \end{aligned}$$

- Generate output by summing inputs weighed by score

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

Self-Attention for Transformers

- Transformers change the way in which different inputs contribute to the output of the self-attention network
- How each input token can act in the attention process:
  - as Query: current focus of attention
  - as Key: preceding input being compared to the query
  - as Value: used to compute the output
- Transformers use weight matrices W^Q, W^K, W^V (all $\in \mathbb{R}^{d \times d}$) to project inputs to these roles

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i; \quad \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i; \quad \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

- Using projections, scoring becomes

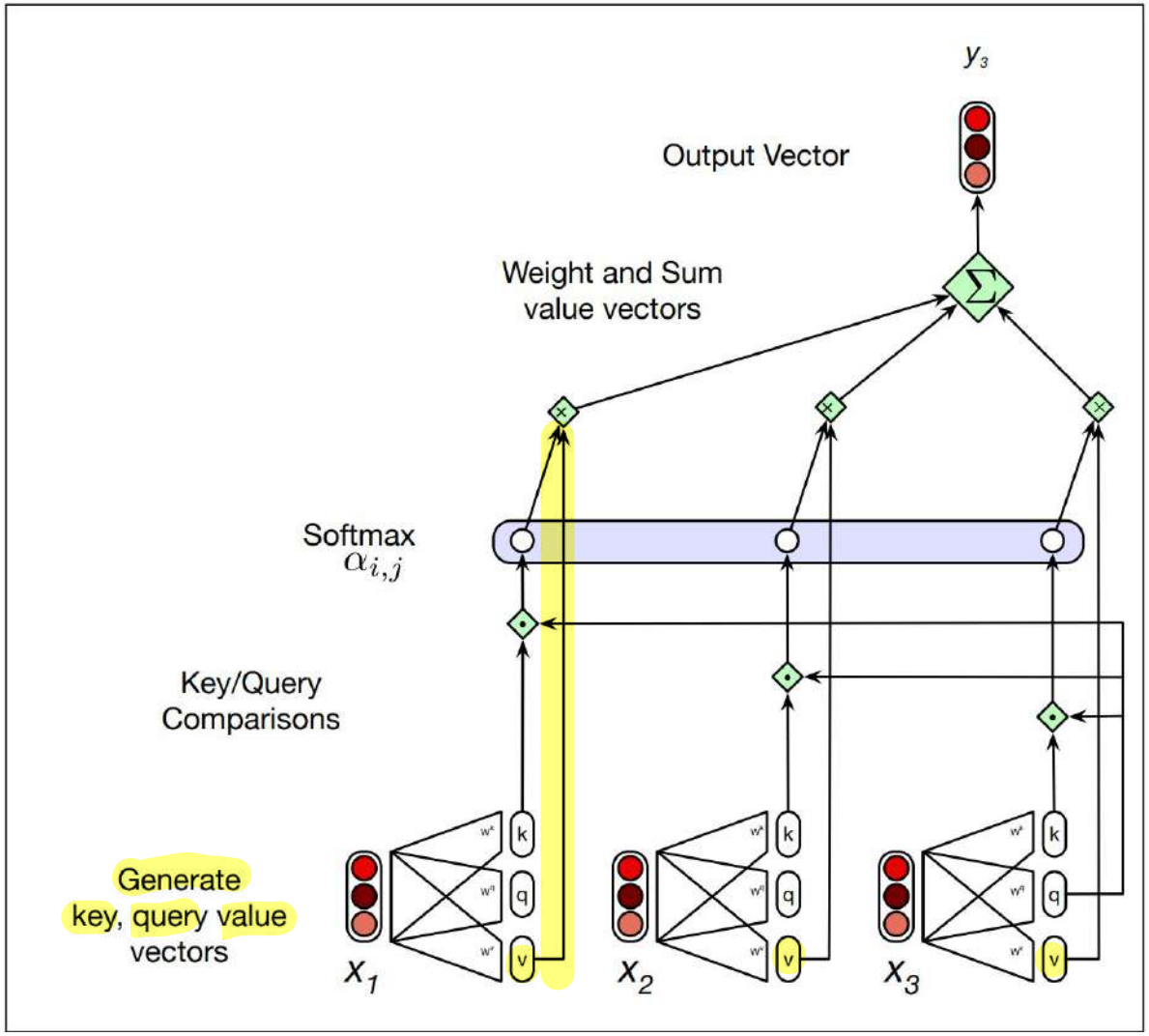
$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$$

- Output calculation is done via the values

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij}\mathbf{v}_j$$

Self-Attention for Transformers

- Transformer-style self-attention computation for a single output y_3



Properties of Self-Attention

- Since dot products can become too large, they are instead scaled by the square root of the dimensionality of the key vector

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

- Key property of this computation:
  - Each output y_i is computed independently
  - Can be further sped up through matrix multiplication (**X, Q, K, V** all $\in \mathbb{R}^{N \times d}$)

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}}; \quad \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}}; \quad \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

$$\textit{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathsf{T}}}{\sqrt{d_k}}\right)\mathbf{V}$$

- Note: Attention is quadratic w.r.t. length of the input

Masking

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

- Problem for decoder style applications (e.g. language modelling): via $\mathbf{QK}^T$ each query would be compared with each key, including those that follow the query
- → we see what we are predicting → solution: masking

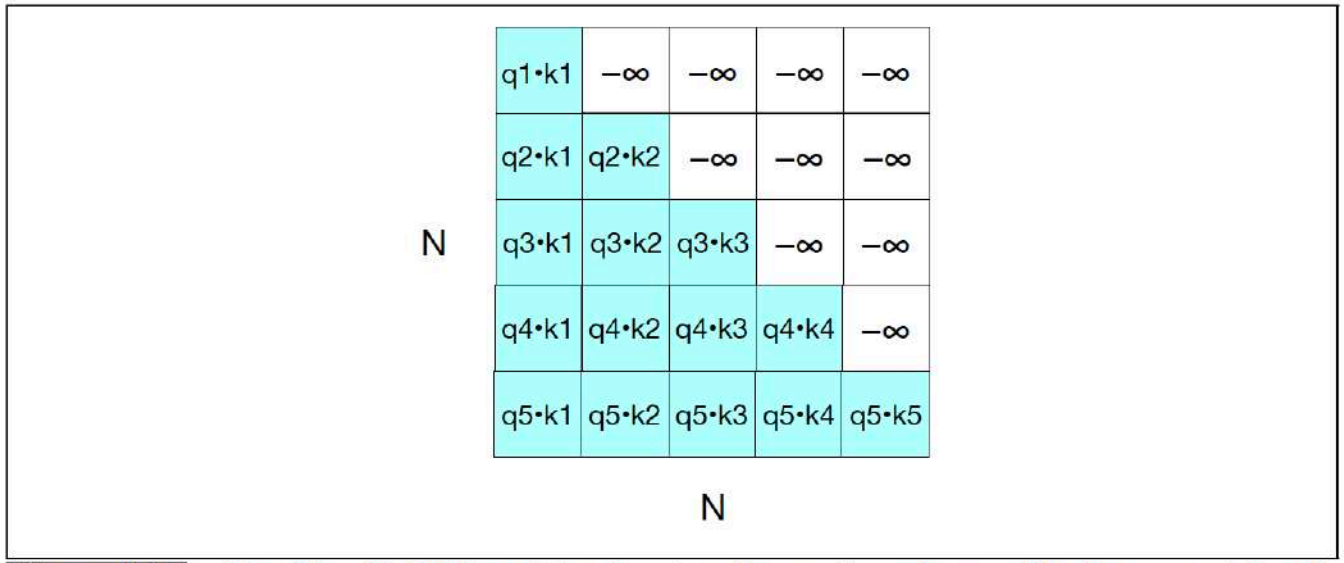
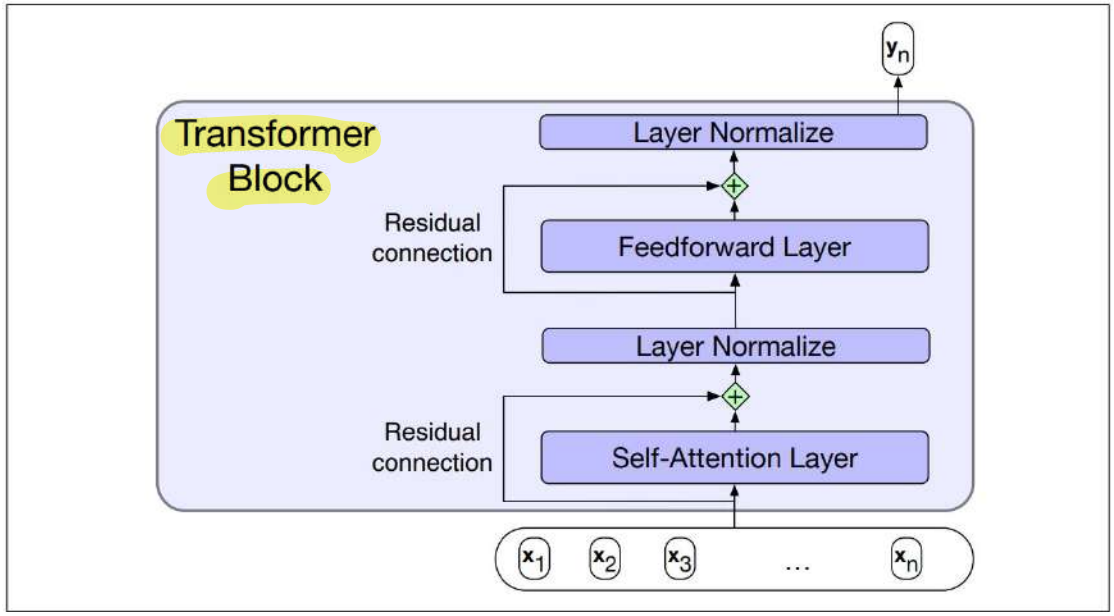


Figure 9.17 The $N \times N$ $\mathbf{QT}^T$ matrix showing the $q_i \cdot k_j$ values, with the upper-triangle portion of the comparisons matrix zeroed out (set to $-\infty$, which the softmax will turn to zero).

Transformer Blocks

- Basis of the transformer block: Self-attention layer
- Additional features:
  - Feedforward layers
  - Residual connections
  - Normalizing layers



- Transformer blocks have identical input and output dimensions, meaning they can be stacked

Layer Norm

- Layer Norms are used to improve training performance
- Keeps values in hidden layers within a specific range to facilitate gradient-based training
- Based on standard z-score via computing mean μ and std deviation σ

$$\mu = \frac{1}{d_h}\sum_{i=1}^{d_h} x_i$$

$$\sigma = \sqrt{\frac{1}{d_h}\sum_{i=1}^{d_h}(x_i-\mu)^2}$$

- Component normalization $\hat{\mathbf{x}} = \frac{(\mathbf{x}-\mu)}{\sigma}$

- Optional: Learnable parameters for gain and offset

$$LayerNorm = \gamma\,\hat{\mathbf{x}} + \beta$$

Multihead Attention

- Words can relate to each other in different ways at the same time
- hard to learn this with a single transformer block

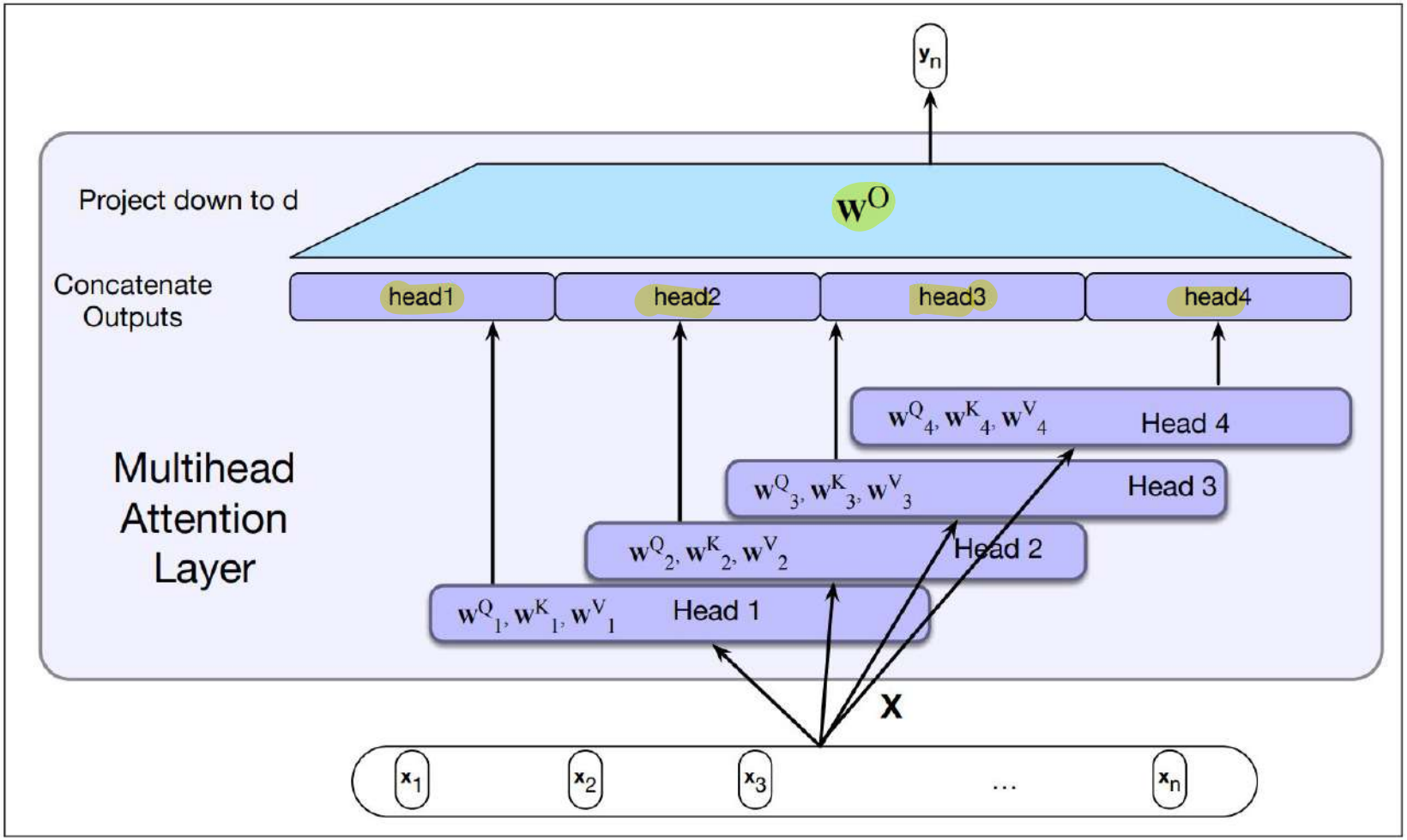
- Solution:
  - Multihead self-attention layers
  - each layer features a distinct set of parameters and weights
  - input is copied across all heads
  - computed in parallel at the same depth of the model
  - at the end of a block reduced back to the original dimensionality by concatenation and linear projection via weight matrix W^0

$$MultiHeadAttn(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W}^O$$

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_i^Q \; ; \; \mathbf{K} = \mathbf{X}\mathbf{W}_i^K \; ; \; \mathbf{V} = \mathbf{X}\mathbf{W}_i^V$$

$$\mathbf{head}_i = SelfAttention(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

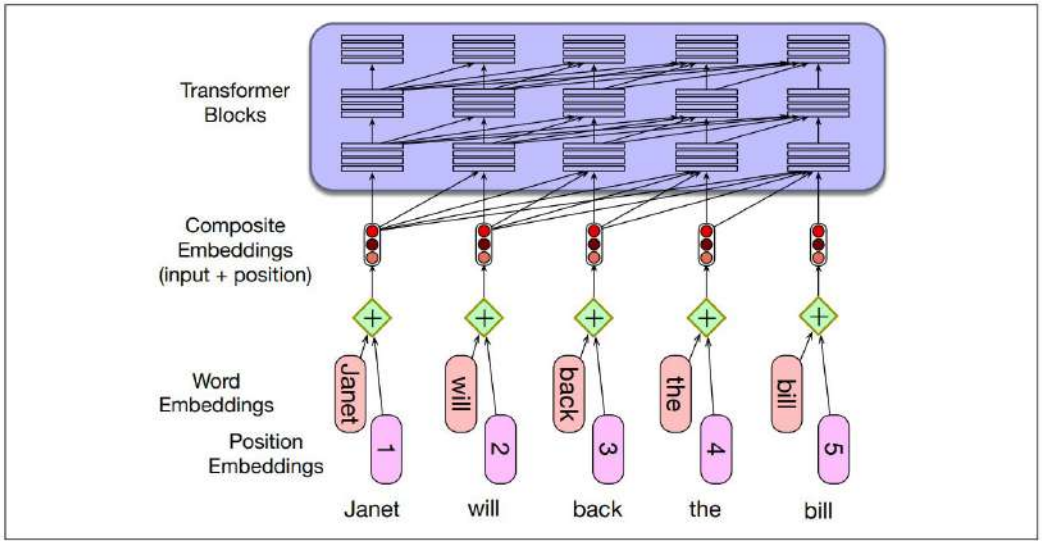
Multihead Attention



Positional Encodings for Transformers

- Positions of tokens are built-in with RNNs
- This is not the case with transformers
- Problem: No notion of relative or absolute position in a sequence

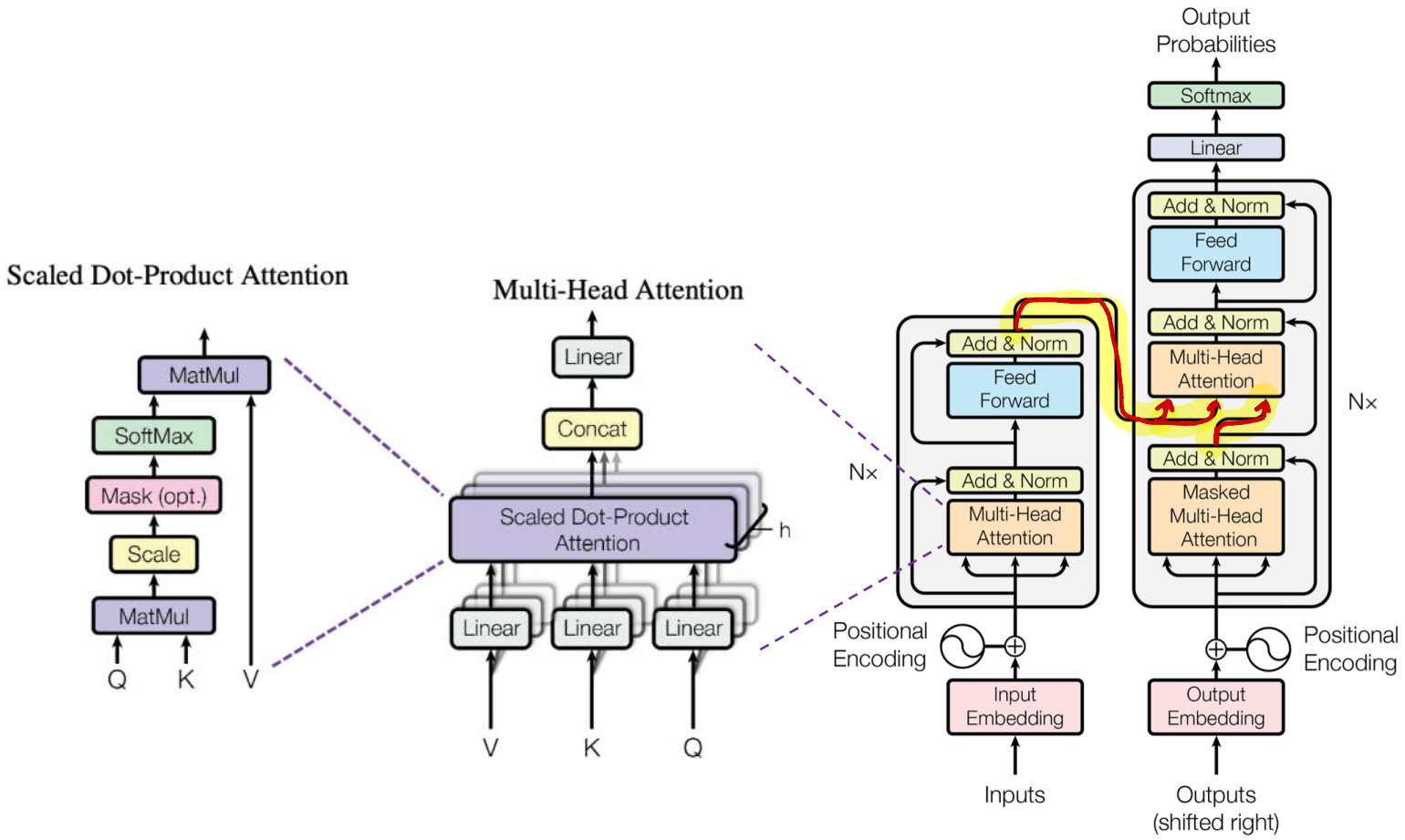
- Solution: Modify input embeddings with positional encodings
- Example: Randomly initialized, trainable positional encodings
  - Either use absolute positions or a mapping to relative positions



↳ normally use trigonometric functions

The Transformer

- Full schema of the original Transformer architecture (Vaswani et al., 2017) for NMT (seq2seq / Encoder-Decoder)



Transformers as Language Models

- Using transformers we can train a language model the same way as with RNNs:
  - Semi-supervised learning
  - Training corpus of plain text
  - Predicting the next word using teacher forcing
  - Compute loss via cross-entropy over the sequence

- Advantages:
  - Calculations are no longer serial
  - Parallelization possible!
  - Resulting model can be evaluated via perplexity
  - New text can be autoregressively generated

- Other possible tasks: Text completion, question answering

Contextual Generation

Contextual generation: model has access to full priming context & all own subsequently generated outputs so far

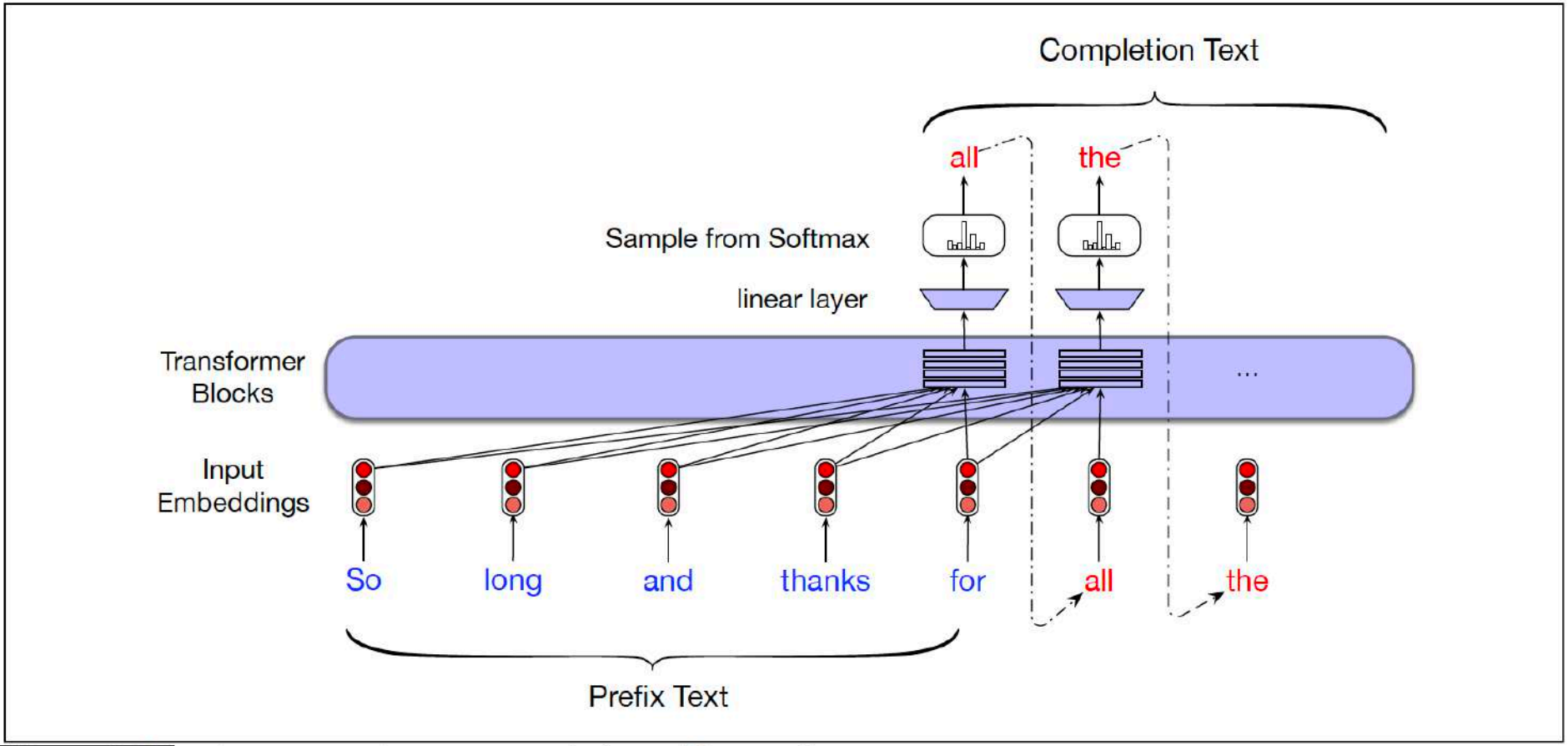
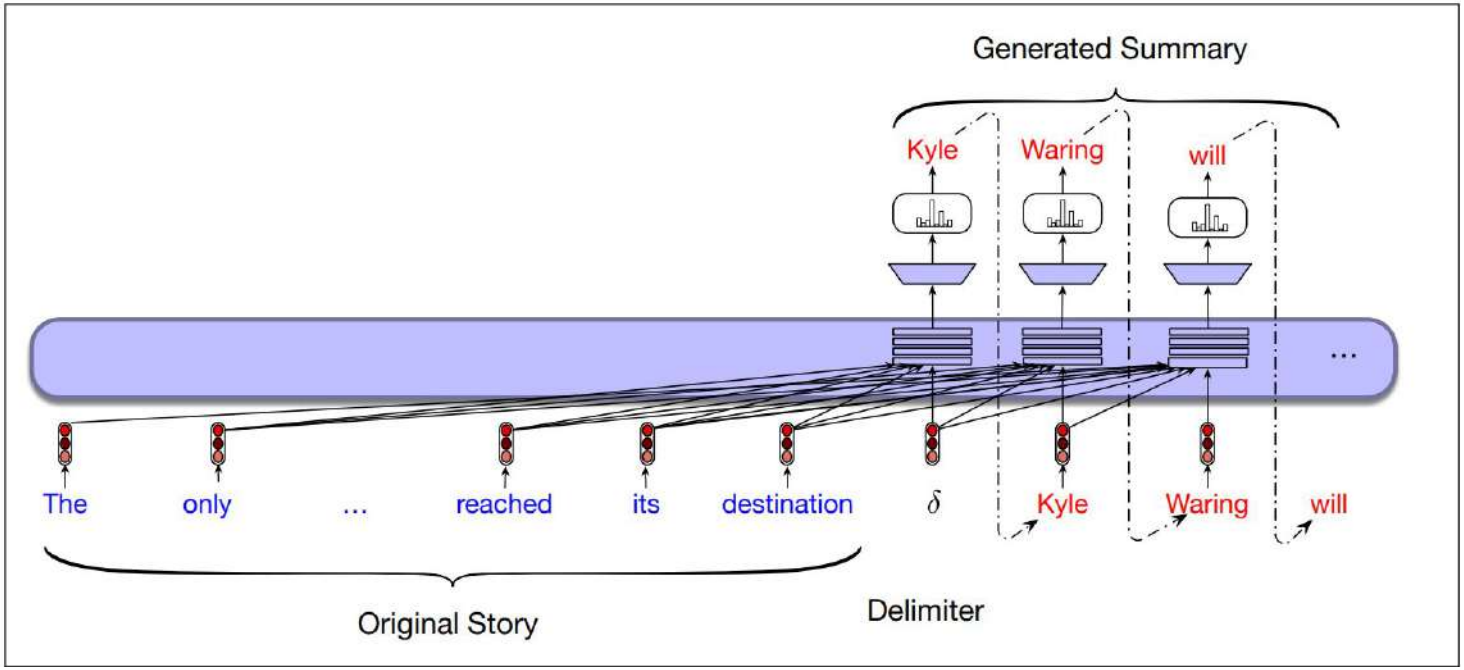


Figure 9.22 Autoregressive text completion with transformers.

Text Summarization using Transformers

- Autoregressive models can be trained for summarization tasks
- Process:
  - Append full length text with its summary, separated by a unique marker
  - Train an autoregressive language model using teacher forcing



Text Summarization using Transformers

Original Article

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For $89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says.

But not if you live in New England or surrounding states. "We will not ship snow to any states in the northeast!" says Waring's website, ShipSnowYo.com. "We're in the business of expunging snow!"

His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. His business slogan: "Our nightmare is your dream!" At first, ShipSnowYo sold snow packed into empty 16.9-ounce water bottles for $19.99, but the snow usually melted before it reached its destination...

Summary

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

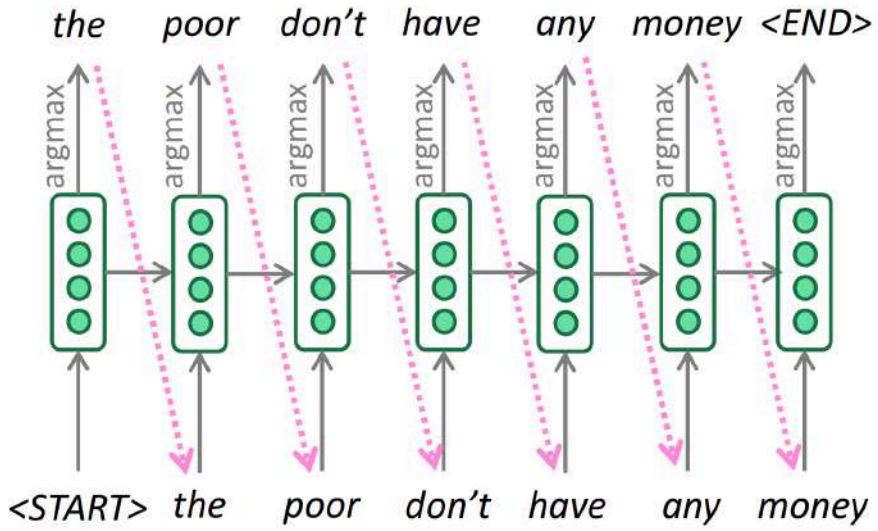
Applying Transformers to Other Tasks

- Other possible transformer tasks include:
  - Sequence labeling
  - Sequence classification

- Usual way:
  - No direct training of the transformer on the task
  - Use pretrained transformer
  - Add feedforward layer on top to finetune on the task

Decoding strategies: Problems of Greedy Decoding

- decoder uses greedy decoding: take most probable word at each step:

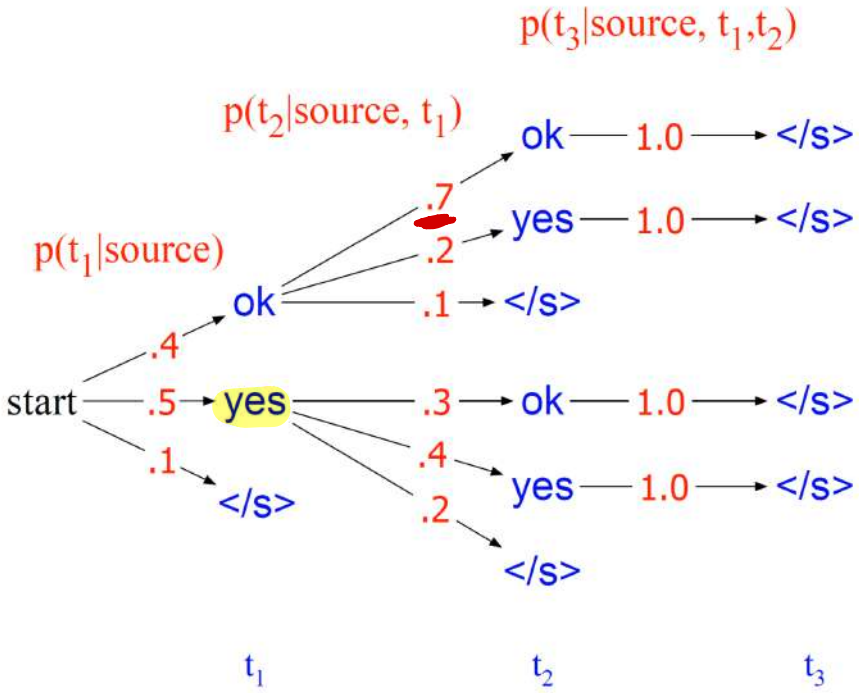


$$\hat{y}_t = \text{argmax}_{w \in V} P(w|x, y_1 \ldots y_{t-1})$$

- problem: no way to undo decisions:
  - *les pauvres sont démunis (the poor don't have any money)*
  - → *the* ____
  - → *the poor* ____
  - → *the poor are* ____

Decoding strategies: Problems of Greedy Decoding

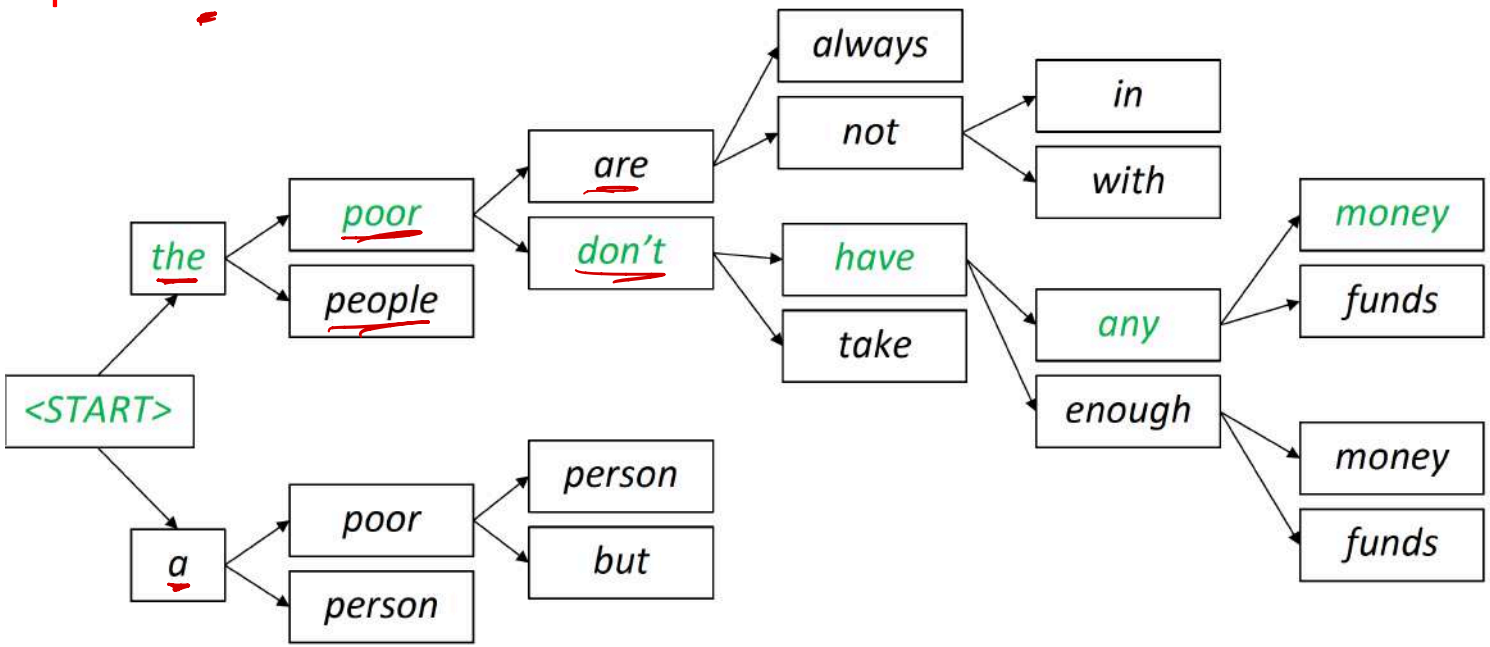
- Search tree example:
  - Sequence with highest probability: *"okok</s>"* with *p=.4*.7*1.0*
  - But: greedy decoding chooses *yes* as first word ☹



Decoding Strategies: Beam Search Decoding

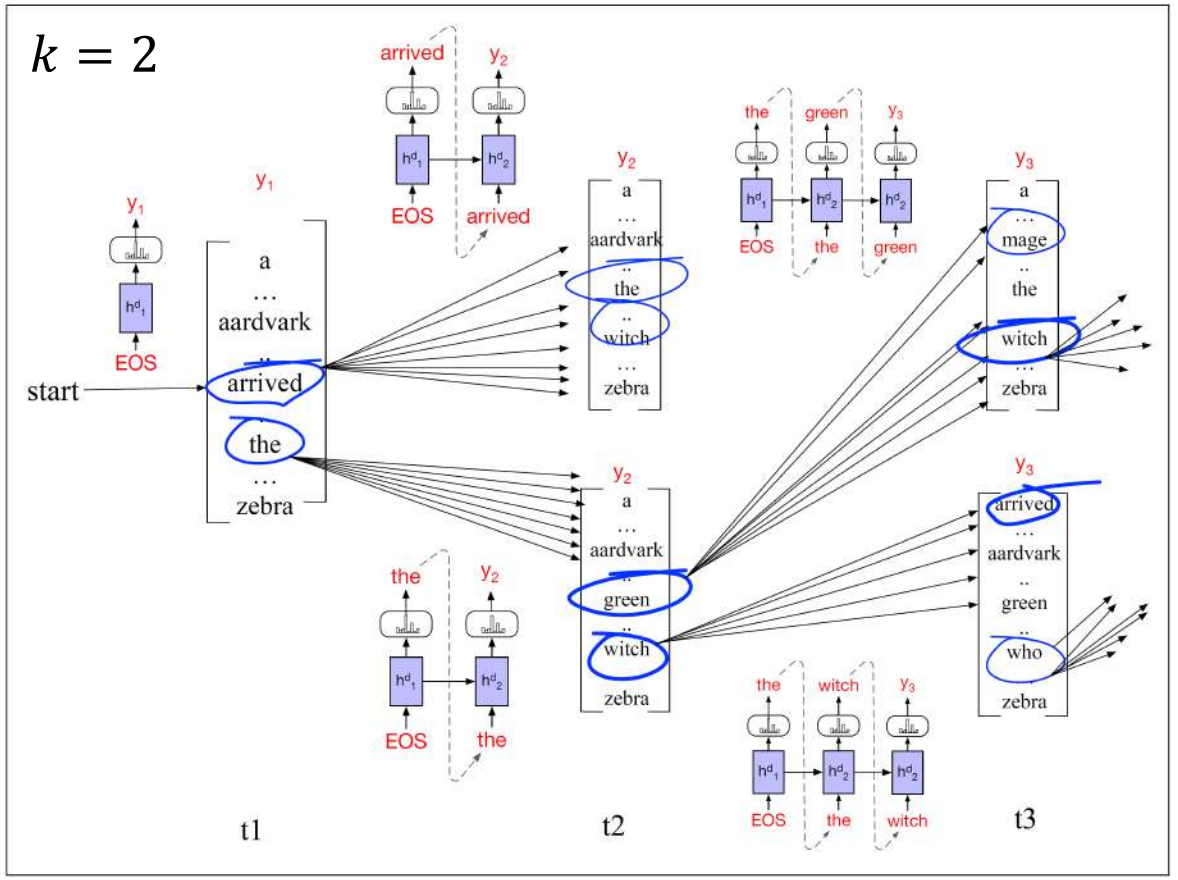
- → use Beam Search: on each step of decoder, keep track of the k most probable partial translations
  - k: beam size (in practice around 5 to 10)
  - not guaranteed to find optimal solution, but much more efficient!

- example: k=2:



Decoding Strategies: Beam Search Decoding

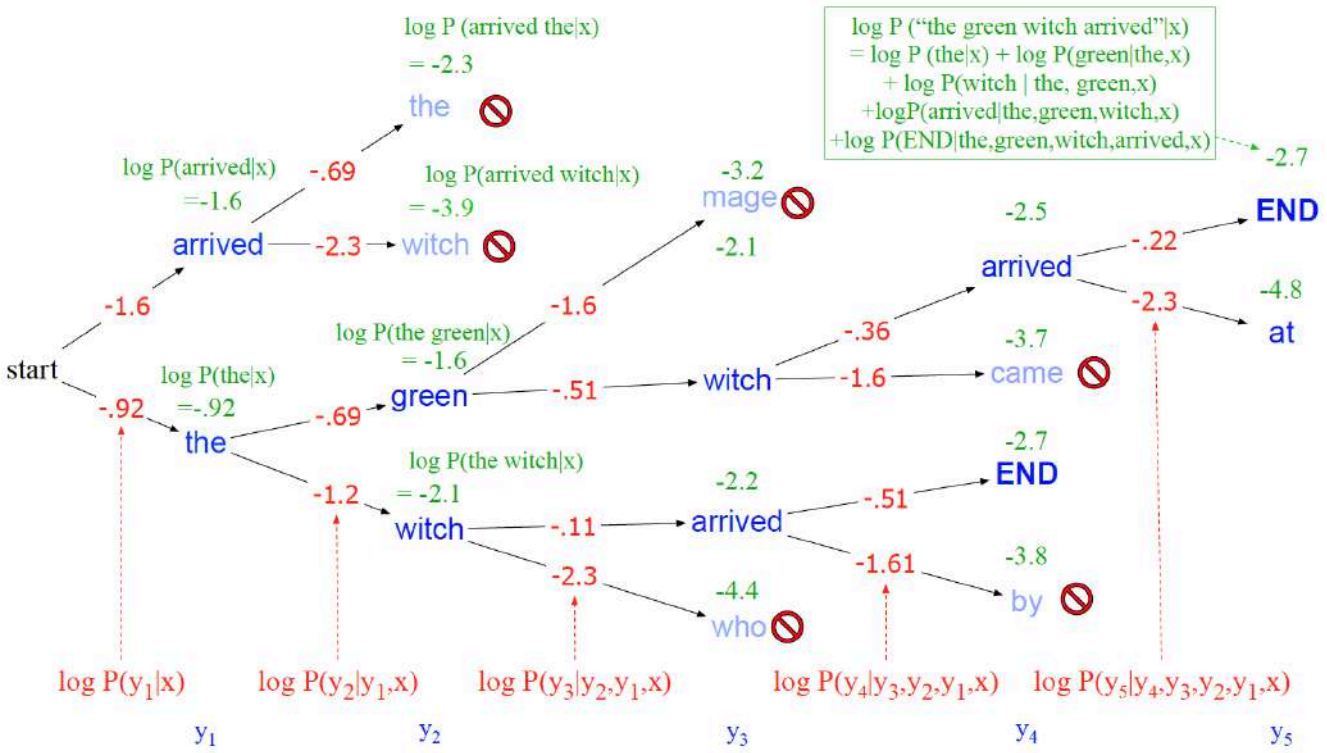
- First decoding step: Softmax over entire vocabulary, select *k*-best options
  → *k* initial words called **hypotheses**
  → parameter *k* called **beam width**
- Subsequent steps: Compute *V* possible extensions by passing through decoder
  → Rank *k*V* new hypotheses and prune to *k*-best
- Hypothesis complete when </s> produced



Decoding strategies: Beam Search

- Ranking hypotheses → Calculate probability of token given existing sequence $P(y_i|x, y_{<i})$

$$
\begin{aligned}
score(y) &= \log P(y|x) \\
&= \log\left(P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x)\ldots P(y_t|y_1,\ldots,y_{t-1},x)\right) \\
&= \sum_{i=1}^{t} \log P(y_i|y_1,\ldots,y_{i-1},x)
\end{aligned}
$$



Decoding strategies: Beam Search

- Generate hypothesis until </s> produced
  → Hypotheses with different length

- Problem: Generally lower probability for longer strings
  → length normalization

$$score(y) = -\log P(y|x) = \frac{1}{T} \sum_{i=1}^{t} -\log P(y_i|y_1, \ldots, y_{i-1}, x)$$

Decoding strategies: Beam search algorithm

```
function BEAMDECODE(c, beam_width) returns best paths

  y_0, h_0 ← 0
  path ← ()
  complete_paths ← ()
  state ← (c, y_0, h_0, path)          ;initial state
  frontier ← ⟨state⟩                   ;initial frontier

  while frontier contains incomplete paths and beamwidth > 0
    extended_frontier ← ⟨⟩
    for each state ∈ frontier do
      y ← DECODE(state)
      for each word i ∈ Vocabulary do
        successor ← NEWSTATE(state, i, y_i)
        extended_frontier ← ADDTOBEAM(successor, extended_frontier,
                                      beam_width)

    for each state in extended_frontier do
      if state is complete do
        complete_paths ← APPEND(complete_paths, state)
        extended_frontier ← REMOVE(extended_frontier, state)
        beam_width ← beam_width - 1
    frontier ← extended_frontier

  return completed_paths

function NEWSTATE(state, word, word_prob) returns new state

function ADDTOBEAM(state, frontier, width) returns updated frontier

  if LENGTH(frontier) < width then
    frontier ← INSERT(state, frontier)
  else if SCORE(state) > SCORE(WORSTOF(frontier))
    frontier ← REMOVE(WORSTOF(frontier))
    frontier ← INSERT(state, frontier)
  return frontier
```

Bibliography

(1) Dan Jurafsky and James Martin: Speech and Language Processing (3rd ed. draft, version Jan, 2023); Online: https://web.stanford.edu/~jurafsky/slp3/ (URL, Oct 2023) (this slideset is especially based on chapters 9 and 10)

(2) Chris Manning et al: "CS224n: Natural Language Processing with Deep Learning", Lecture Materials (slides and links to background reading) http://web.stanford.edu/class/cs224n/ (URL, Oct 2022), 2020

(3) https://www.youtube.com/playlist?list=PLoROMvodv4rOSH4v6133s9LFPRHjEmbmJ (URL, Oct 2022) (in [2])

(4) Rico Sennrich (University of Edinburgh): Neural Machine Translation: Breaking the Performance Plateau, Talk July 2016; http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf (URL, Aug 2018) (in [2])

(5) Richard Socher et al: "CS224n: Natural Language Processing with Deep Learning", Lecture Materials (slides and links to background reading) http://web.stanford.edu/class/cs224n/ (URL, May 2018), 2018

Recommendations for Studying

- minimal approach:

  work with the slides and understand their contents! Think beyond instead of merely memorizing the contents

- standard approach:

  minimal approach + read the corresponding pages in Jurafsky [1]

- interested students

  == standard approach + read the additional papers in bibliography. Download an available IPython implementation of Transformers and experiment with it, visualizing self attention scores.