Self-Attention Networks: Transformers

Reference:
- D. Jurafsky and J. Martin, “Speech and Language Processing”

Motivation

- For RNNs such as LSTMs, passing information forward through an extended series of recurrent connections leads to a loss of relevant information and to difficulties in training
- Inherently sequential nature of recurrent networks *inhibits* the use of parallel computational resources
- Transformers – an approach to sequence processing that eliminates recurrent connections and returns to architectures reminiscent of the fully connected networks.

Basic Architecture

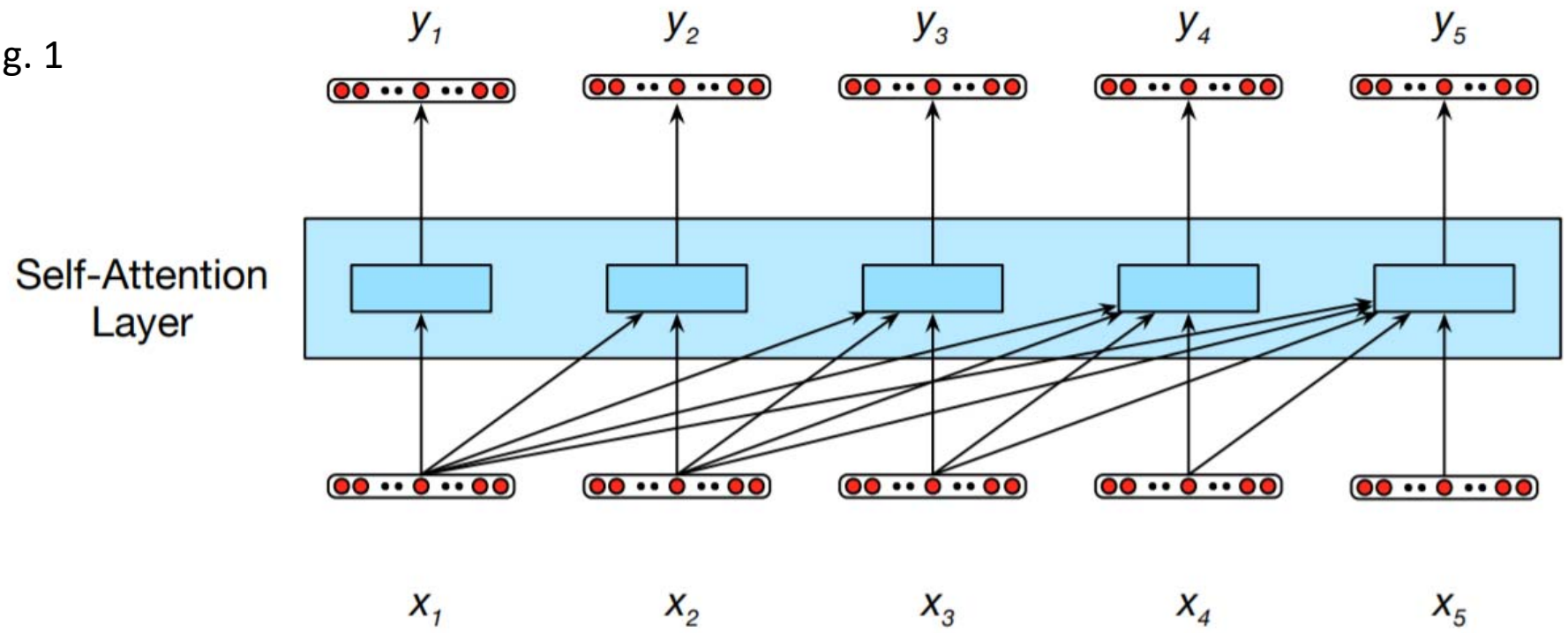
- Map sequence of input vectors $(x_1, \cdots, x_n)$ to sequences of output vectors $(y_1, \cdots, y_n)$ of the same length
- Made up of stacks of network layers consisting of simple linear layers, feedforward networks, and custom connections around them
- Use of self-attention layers – key innovation of Transformers
- Self-attention allows a network to directly extract and use information from arbitrarily large contexts without the need to pass it through intermediate recurrent connections as in RNNs

Self-Attention Layers

- Map input sequences $(x_1, \cdots, x_n)$ to output sequences of the same length $(y_1, \cdots, y_n)$, as with the overall Transformer
- When processing each item in the input, have access to all of the inputs up to and including the one under consideration, but no access to information about inputs beyond the current one
  - Ensures we can use this approach to create language models and use them for autoregressive generation
- Computation performed for each item is independent of all the other computations
  - Means that we can easily parallelize both forward inference and training of such models

Self-Attention Layers

Fig. 1



Information flow in a causal (or masked) self-attention model. In processing each element of the sequence, the model attends to all the inputs up to, and including, the current one. Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

Self-Attention is an Attention-Based Approach

- An attention-based approach – to compare an item of interest to a collection of other items in way that reveals their relevance in the current context
- Self-attention – the set of comparisons are to other elements within a given sequence
- The result of these comparisons is then used to compute an output for the current input
- Returning to Fig. 1, the computation of y_3 is based on a set of comparisons between the input x_3 and its preceding elements x_1 and x_2, and to x_3 itself

Self-Attention – the Simplest Form

- The simplest form of comparison between elements in a self-attention layer is a dot product
- Referring to the result of these comparisons as scores

$$score(x_i, x_j) = x_i \cdot x_j$$

  - Ranging from $-\infty$ to ∞, the larger the value the more similar the vectors that are being compared
- The first step in computing y_3 would be to compute three scores: $x_3 \cdot x_1$, $x_3 \cdot x_2$ and $x_3 \cdot x_3$

Self-Attention – the Simplest Form

- To make effective use of these scores, normalize them with a softmax to create a vector of weights, α_{ij}, that indicates the proportional relevance of each input to the input element i that is the current focus of attention

$$\alpha_{ij} = \text{softmax}\left(score\left(x_i, x_j\right)\right) \; \forall j \leq i$$

$$= \frac{\exp\left(score\left(x_i, x_j\right)\right)}{\sum_{k=1}^{i} \exp\left(score\left(x_i, x_k\right)\right)} \; \forall j \leq i$$

- Given the proportional scores in α, we may generate an output value y_i by taking the sum of the inputs seen so far, weighted by their respective α value

$$y_i = \sum_{j \leq i} \alpha_{ij} x_j$$

Self-Attention – the Simplest Form

- Steps embodied above represent the core of an attention-based approach:
  - A set of comparisons to relevant items in some context,
  - A normalization of those scores to provide a probability distribution, followed by a weighted sum using this distribution.
  - The output y is the result of this straightforward computation over the inputs.
- Unfortunately, this simple mechanism provides no opportunity for learning, everything is directly based on the original input values x.

Self-Attention with Additional Parameters

- To allow for this kind of learning, Transformers include additional parameters in the form of a set of weight matrices that operate over the input embeddings.
- To motivate these new parameters, consider the different roles that each input embedding plays during the course of the attention process.
  - As the *current attention focus* when being compared to all of the other preceding inputs – a **query**
  - In its role as a *preceding input* being compared to the current focus of attention – a **key**
  - And finally, used to compute the *output* for the current focus of attention – a **value**

Self-Attention with Additional Parameters

- To capture the different roles that input embeddings play in each of these steps, Transformers introduce three sets of weights which we'll call W^Q, W^K, and W^V
- These weights will be used to compute linear transformations of each input x with the resulting values being used in their respective roles in subsequent calculations

$$q_i = W^Q x_i;\ k_i = W^K x_i;\ v_i = W^V x_i$$

  - Given input embeddings of size d_m, the dimensionality of these matrices are $d_q \times d_m$, $d_k \times d_m$ and $d_v \times d_m$ respectively
  - In the original Transformer work (Vaswani et al., 2017), d_m was 1024 and 64 for d_k, d_q and d_v

Self-Attention with Additional Parameters

- Given these projections, the score between a current focus of attention, x_i and an element in the preceding context, x_j consists of a dot product between its query vector q_i and the preceding elements key vectors k_j

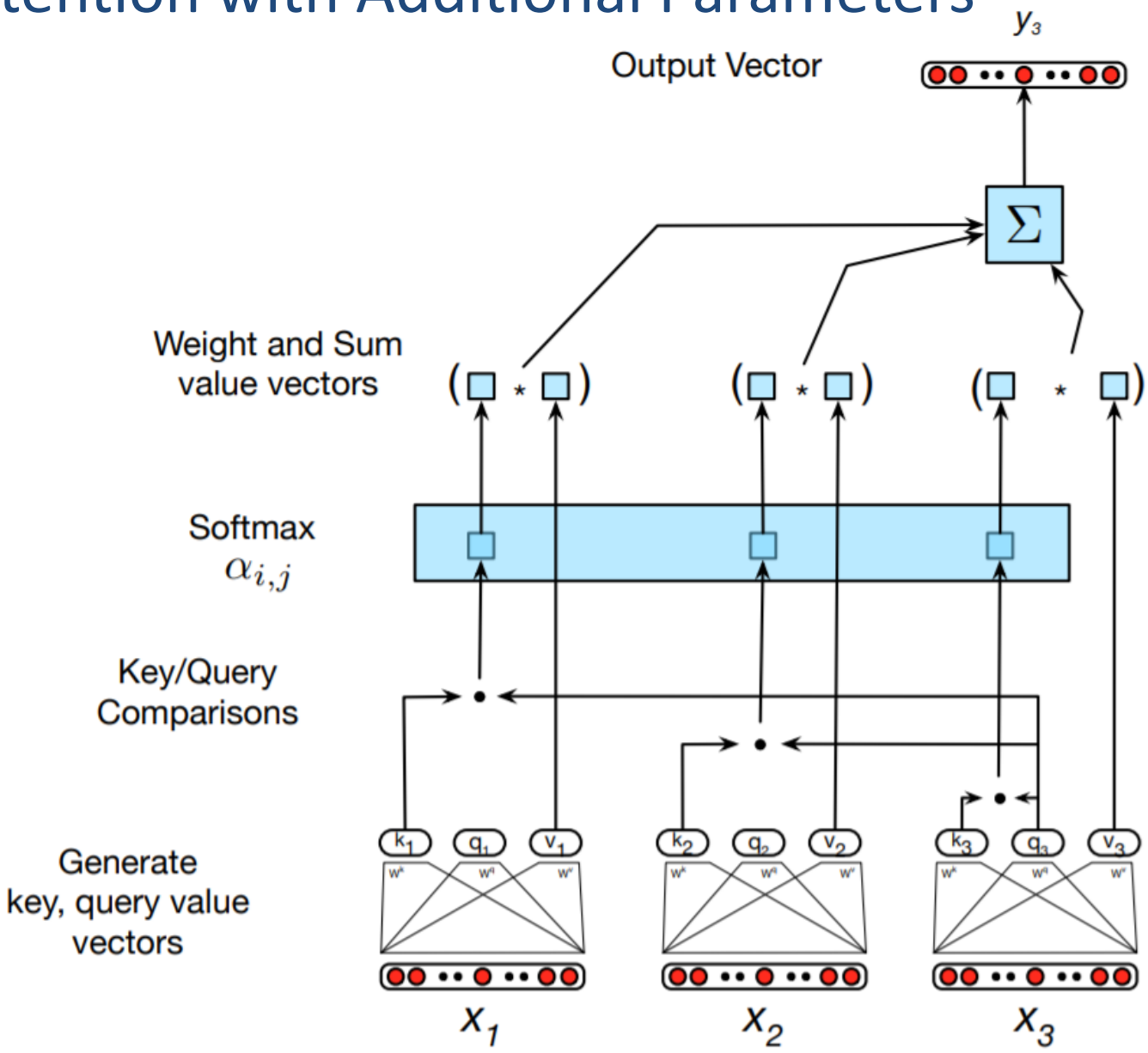
$$score(x_i, x_j) = q_i \cdot k_j$$

- The ensuing softmax calculation resulting in α_{ij} remains the same
- The output calculation for y_i is now based on a weighted sum over the value vectors v

$$y_i = \sum_{j \leq i} \alpha_{ij} v_j$$

Self-Attention with Additional Parameters

Fig. 2



Calculation of the value of the third element of a sequence using causal self-attention.

Self-Attention – Scaled Dot-Product

- The result of dot product can be an arbitrarily large (positive or negative) value.
- In computing α_{ij}, exponentiating such large values can lead to numerical issues and to an effective loss of gradients during training.
- A scaled dot-product approach divides the result of the dot product by a factor related to the size of the embeddings before passing them through the softmax
- A typical approach is to divide the dot product by the square root of the dimensionality of the query and key vectors

$$score(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

Self-Attention – Parallelism

- By packing the input embeddings into a single matrix, the entire process can be parallelized by taking advantage of efficient matrix multiplication routines

$$Q = W^Q X;\ K = W^K X;\ V = W^V X$$

- The entire self-attention step for an entire sequence

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

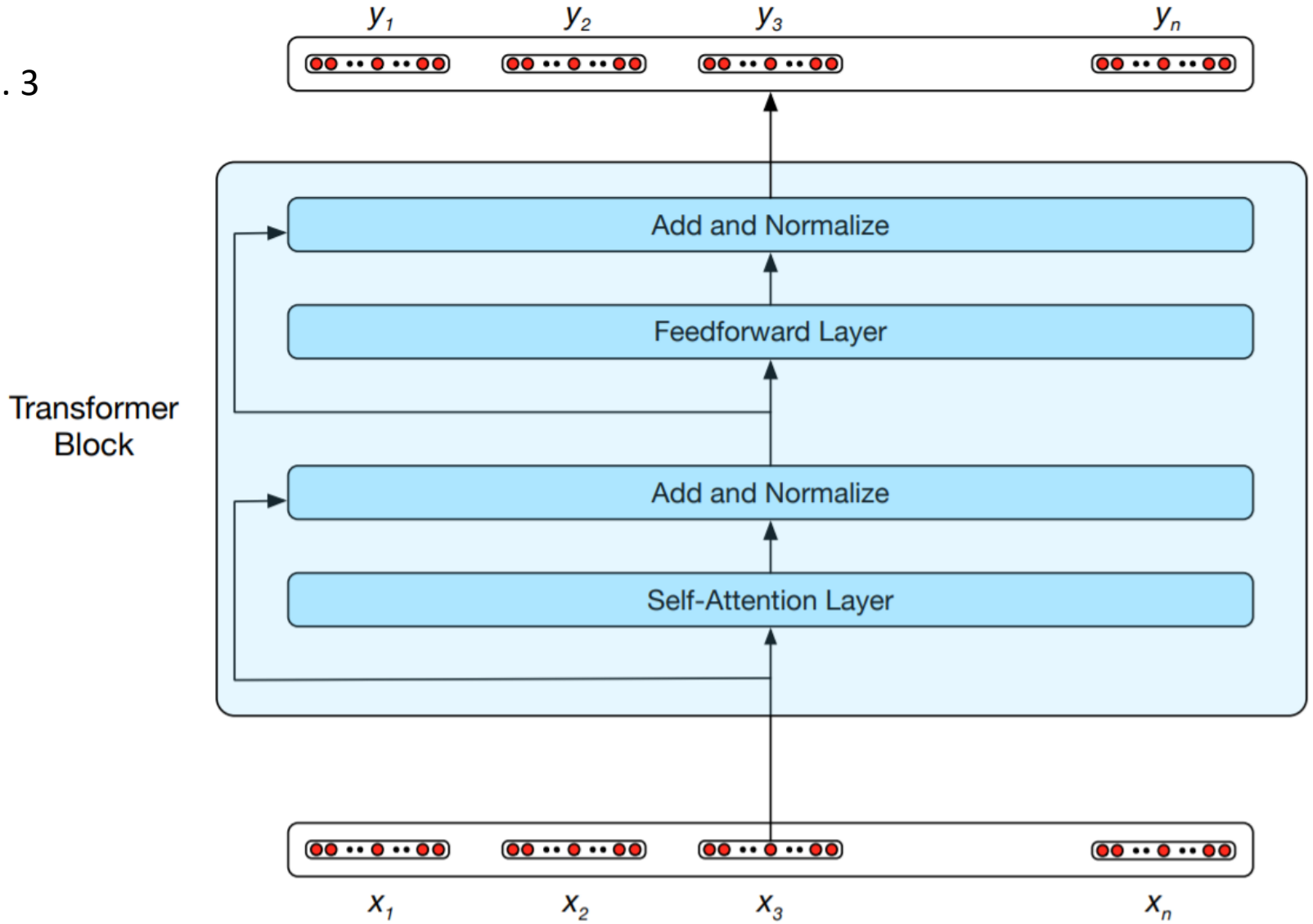
  - The calculation of the comparisons in QK^T results in a score for each query value to every key value, including even those that follow the query
  - To fix this, the elements in the upper-triangular portion of the comparisons matrix are zeroed out (set to $-\infty$)

Transformer Blocks

- In addition to the self-attention layer, a Transformer block includes additional feedforward layers, residual connections, and normalizing layers.
- These blocks can then be stacked just as was the case for stacked RNNs.

Transformer Blocks

Fig. 3



A typical transformer block consisting of a single attention layer followed by a fully-connected feedforward layer with residual connections and layer normalizations following each.

Multihead Attention

- The different words in a sentence can relate to each other in many different ways simultaneously, e.g. distinct syntactic, semantic, and discourse relationships between verbs and their arguments in a sentence.
  - It'd be difficult for a single transformer block to learn to capture all of the different kinds of parallel relations among its inputs.
- Transformers address this issue with multihead self-attention layers
  - Sets of self-attention layers, called heads, that reside in parallel layers at the same depth in a model, each with its own set of parameters.
- Given these distinct sets of parameters, each head can learn different aspects of the relationships that exist among inputs at the same level of abstraction.

Multihead Attention

- To implement this notion, each head, i, in a self-attention layer is provided with its own set of key, query and value matrices: W_i^K, W_i^Q and W_i^V. These are used to project the inputs to the layer, x_i, separately for each head.
- The output of a multi-head layer with h heads consists of h vectors of the same length.

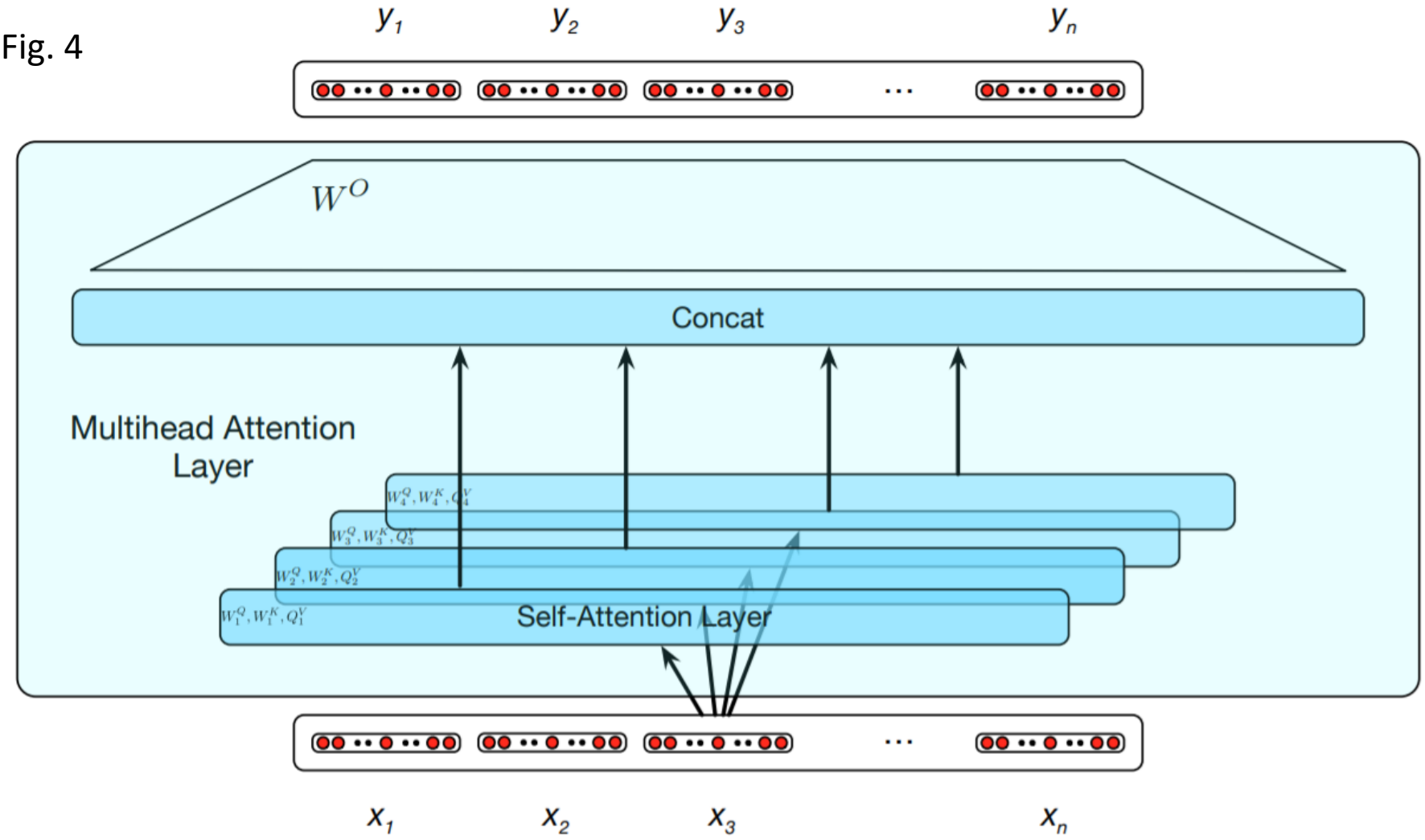
$$head_i = SelfAttention\left(W_i^Q X, W_i^K X, W_i^V X\right)$$

- They are combined and then reduced down to the original input dimension d_m by concatenating the outputs from each head and then using yet another linear projection to reduce it to the original output dimension.

$$MultiHeadAttn(Q, K, V) = W^O(head_1 \oplus head_2 \oplus \cdots \oplus head_h)$$

Multihead Attention

Fig. 4



Multihead self-attention: Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices. The outputs from each of the layers are concatenated and then projected down to d_{model}, thus producing an output of the right size.

Positional Embeddings

- Unlike RNNs, there's nothing that would allow Transformers to make use of information about the relative, or absolute, positions of the elements of an input sequence.
- If you scramble the order of inputs in the attention computation illustrated earlier, you get exactly the same answer.
- To address this issue, Transformer inputs are combined with positional embeddings specific to each position in an input sequence.

Positional Embeddings

A Simple Approach

- Rrandomly initialized embeddings corresponding to each possible input position up to some maximum length, e.g. an embedding for the position 3 just like an embedding for the word fish.
- As with word embeddings, learn these positional embeddings along with other parameters during training.
- To produce an input embedding that captures positional information, add the word embedding for each input to its corresponding positional embedding. This new embedding serves as the input for further processing.
- A potential problem: too few training examples at the outer length limits and poorly trained latter embeddings

Positional Embeddings

Using A Static Function

- Choose a static function that maps an integer inputs to real-valued vectors in a way that captures the inherent relationships among the positions, e.g. the fact that position 4 in an input is more closely related to position 5 than it is to position 17
- A combination of sine and cosine functions with differing frequencies was used in the original Transformer work

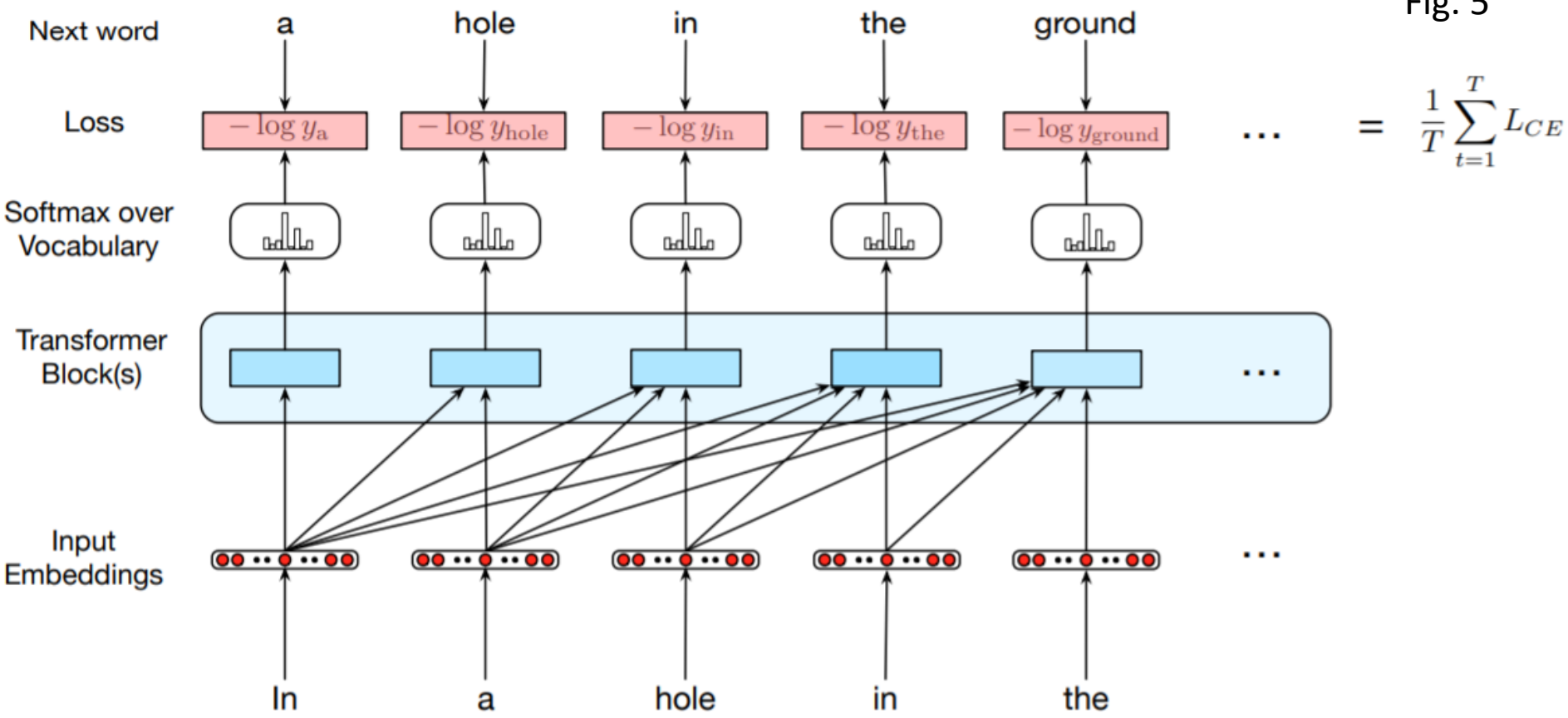
Transformers as Autoregressive Language Models

Let's examine how to deploy them as language models via semi-supervised learning.

- Proceed just as we did with the RNN-based approach: given a training corpus of plain text we'll train a model to predict the next word in a sequence using teacher forcing.
- At each step, given all the preceding words, the final Transformer layer produces an output distribution over the entire vocabulary.
- During training, the probability assigned to the correct word is used to calculate the cross-entropy loss for each item in the sequence.
- As with RNNs, the loss for a training sequence is the average cross-entropy loss over the entire sequence.

Transformers as Autoregressive Language Models

Fig. 5



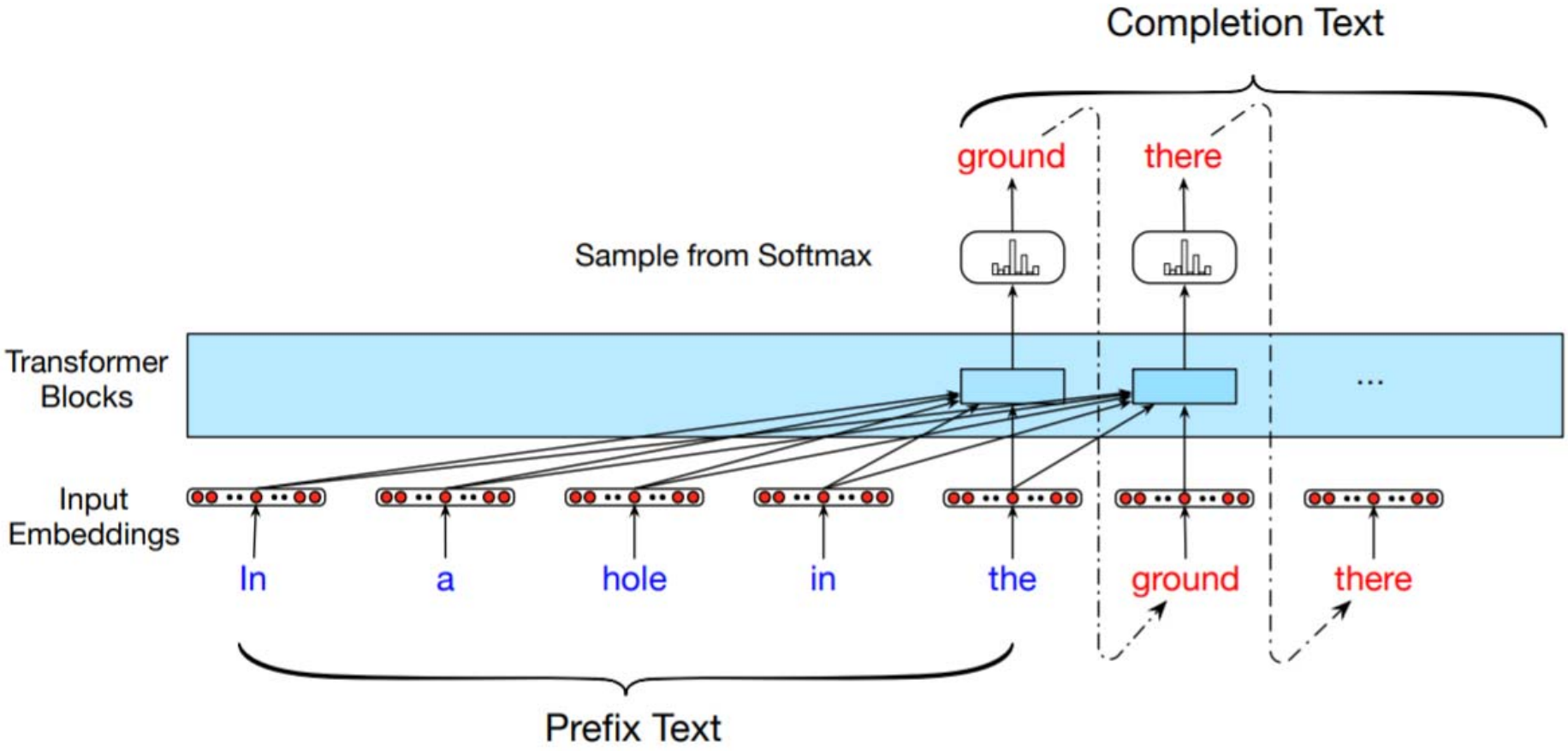
- With Transformers, each training item can be processed in parallel since the output for each element in the sequence is computed separately.
- Once trained, we can compute perplexity of the resulting model, or autoregressively generate novel text just as with RNN-based models.

Contextual Generation

- A simple variation on autoregressive generation that underlies a number of practical applications uses a prior context to prime the autoregressive generation process.
- A standard language model is given the prefix to some text and is asked to generate a possible completion to it.
- As the generation process proceeds, the model has direct access to the priming context as well as to all of its own subsequently generated outputs.
- This ability to incorporate the entirety of the earlier context and generated outputs at each time step is the key to the power of these models.

Contextual Generation

Fig. 6



Autoregressive text completion with Transformers.

Text Summarization

- Text summarization is a practical application of context-based autoregressive generation.
- The task is to take a full-length article and produce an effective summary of it.
- To train a Transformer-based autoregressive model to perform this task, start with a corpus consisting of full-length articles accompanied by their corresponding summaries.

Text Summarization

Fig. 7

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.

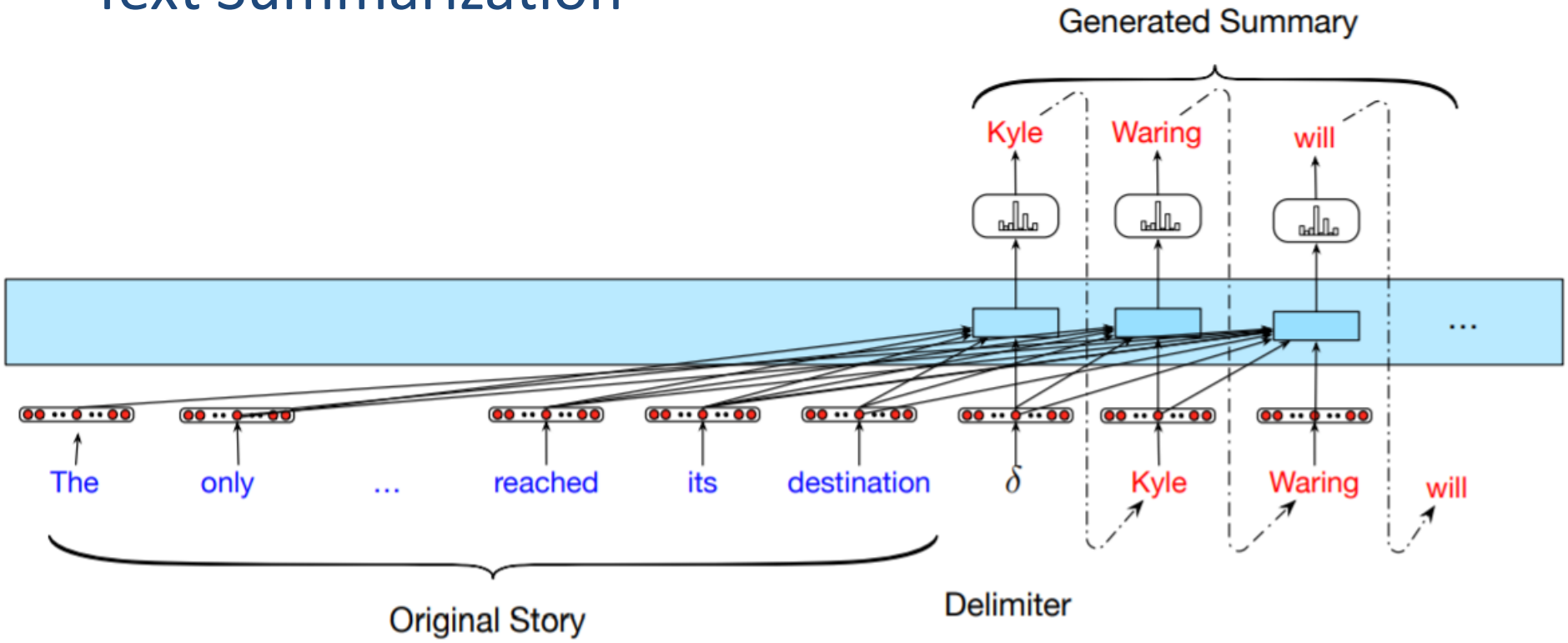
Examples of articles and summaries from the CNN/Daily Mail corpus (Hermann et al., 2015), (Nallapati et al., 2016).

Text Summarization

- A surprisingly effective approach to applying Transformers to summarization is to append a summary to each full-length article in a corpus, with a unique marker separating the two.
- More formally, each article-summary pair $(x_1, \ldots, x_m)$, $(y_1, \ldots, y_n)$ in a training corpus is converted into a single training instance $(x_1, \ldots, x_m, \delta, y_1, \ldots y_n)$ with an overall length of $n + m + 1$.
- These training instances are treated as long sentences and then used to train an autoregressive language model using teacher forcing, exactly as we did earlier.
- Once trained, full articles ending with the special marker are used as the context to prime the generation process to produce a summary.

Text Summarization

Fig. 8



Summarization with Transformers

- The model has access to the original article as well as to the newly generated text throughout the process.
- Variations on this simple scheme are the basis for successful text-to-text applications including machine translation, summarization and question answering.