# Transformer Models

## Motivation

NLP tasks require good text embeddings, which can be learned using neural networks. Before transformers, recurrent neural networks were primarily used.

RNNs suffer from two problems:

- Sequence length / long-range dependencies
  - NLP tasks typically require long-distance dependencies between terms
  - Problem: vanishing gradient over long distances in RNNs
  - Solution: reduce the lengths of paths that signals must traverse
- Training efficiency
  - Parallelized computation within sequences enables large-scale training
  - Problem: recurrent models cannot be parallelized at sequence level (state at timestep $t + 1$ depends on state at $t$), only on batch level
  - Solution: independent computation of each timesteps state

Transformers are efficient (in-sample parallelization) while also handling long-range dependencies (constant path length), enabled by the Attention mechanism.

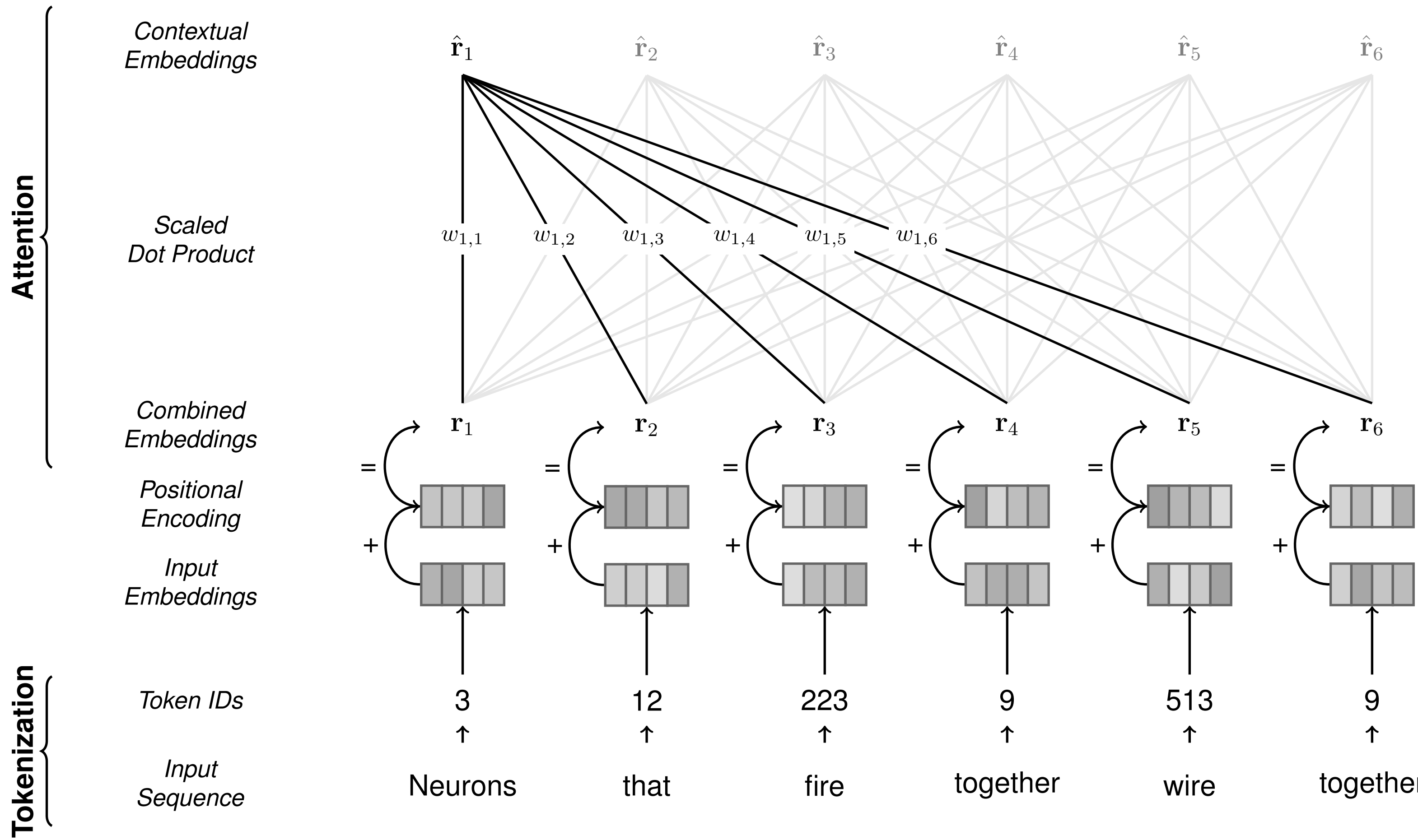
- Goal: allow each embedding to consider the other tokens in the sequence
  - this allows to build contextualized embeddings, i.e., token embeddings that include information about the context around the token
  - starting point is an embedding layer, a simple lookup table that pairs each input token ID with a learned continuous vector (input embedding)
- Idea: represent each token as weighted sum of all tokens in the sequence

$$\hat{\mathbf{r}}_i = \sum_{j=1}^{t} w_{i,j} \cdot \mathbf{r}_j$$

- This requires four building blocks:
  1. a way to decompose strings into tokens ➜ Tokenization
  2. an initial representation of each token ➜ Input Embeddings
  3. a way of representing the order of tokens ➜ Positional Encoding
  4. a way of computing $\hat{\mathbf{r}}_i$ for each token ➜ Attention

# Transformer Models

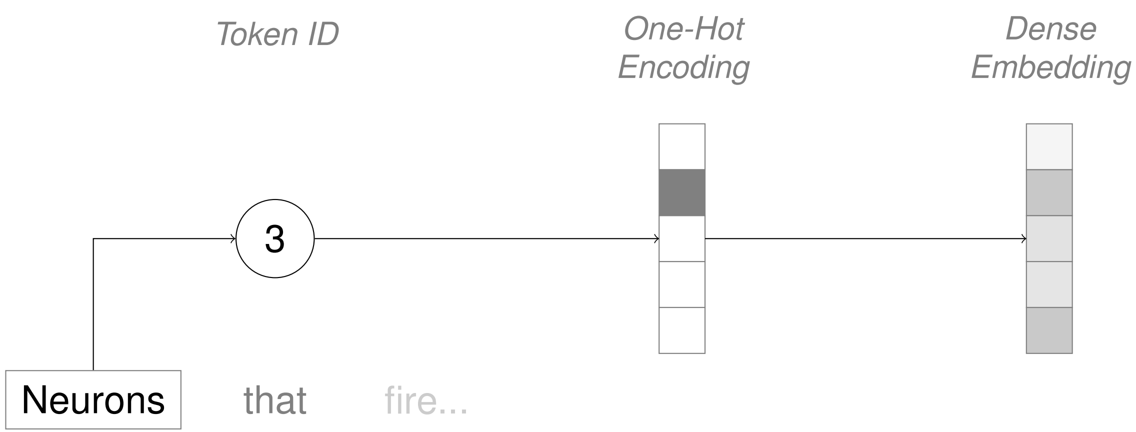
## Contextual Embeddings




**Note**: weights $w$ are not learned directly, but inferred by the attention mechanism from learned projections $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ of the combined embeddings.

# Transformer Models
## Tokenization & Input Embeddings

- The tokenizer splits the input string into a sequence of integers which represent each tokens' index in the vocabulary of the tokenizer
- Input embeddings are formed by projecting the (sparse) one-hot encoded token IDs into a (dense) vector space capturing semantic information
- This projection is learned jointly with the rest of the model



# Transformer Models

## Positional Encoding

- Word order (token position in the sequence) is crucial for language tasks
  - Transformer models lack recurrency, all tokens in the are fed to the network in parallel ➜ position information is lost
  - We need to add some information indicating the word order (position of the word) to the input embeddings ➜ positional encoding
- **Naive approach**: use token indices as positional encodings
  - represent the position of each element in the sequence by its index
  - indices are not bounded and can grow large in magnitude
  - normalized (0-1) indices are incompatible with variable length sequences as these would be normalized differently
- **Better approach**: represent position by a vector where each dimension corresponds to a different sine function evaluated at the current index
  - compatible with long sequences (bounded in magnitude)
  - compatible with variable length sequences (normalized)
  - compatible with arbitrary input embedding dimensionalities

## Remarks

Sinusoidal positional encodings represent position by a vector where each dimension corresponds to the output of a function evaluating the current positions' index. Even dimensions are mapped with a sine function, odd positions are mapped with a cosine function, all of differing frequencies.

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$
$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

Parameters:

- $k$ ➜ Index in the input sequence
- $d$ ➜ Dimensinoiality of positional encoding
- $n$ ➜ Scalar normalization constant (usually $10,000$)
- $i$ ➜ Used for mapping to column indices, $i \in [0 \ldots d/2]$

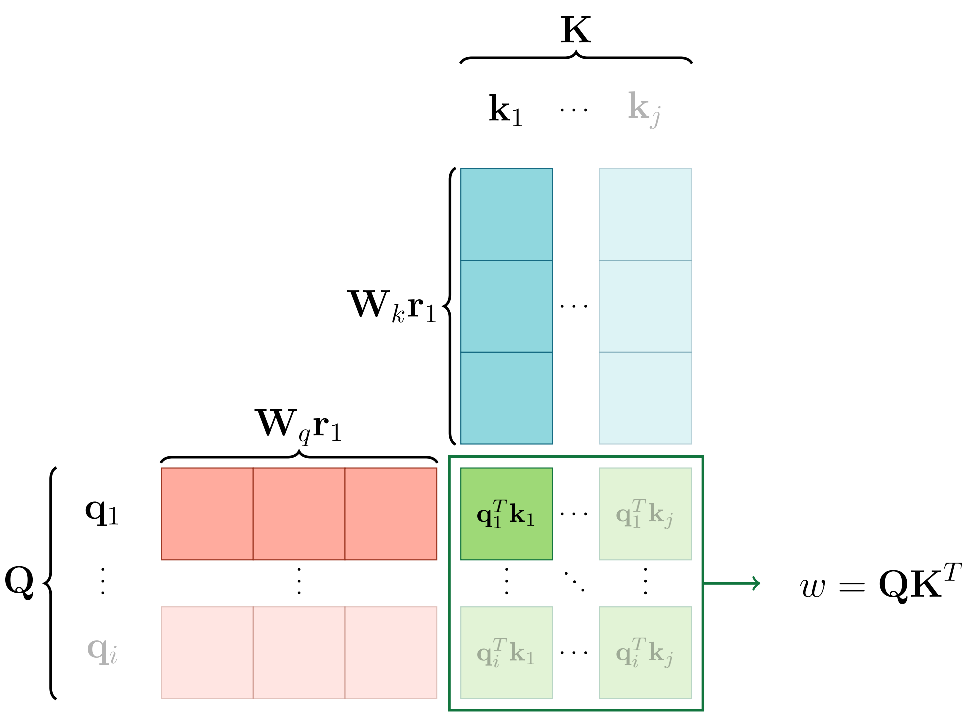
# Transformer Models

## Scaled Dot Product

To determine $w_{i,j}$ for two token representations $\mathbf{r}_i$ and $\mathbf{r}_j$, we:

- compute a query vector $\mathbf{q}_i = \mathbf{W}_q\mathbf{r}_i$ as linear transformation of $\mathbf{r}_i$
- compute a key vector $\mathbf{k}_j = \mathbf{W}_k\mathbf{r}_j$ as linear transformation of $\mathbf{r}_j$
- compute the dot product $w_{i,j} = \mathbf{q}_i^T\mathbf{k}_j$ that indicates token similarity

This is repeated for every token as key and as query, yielding the weight matrix $w$.




- query vectors are stacked into a query matrix $\mathbf{Q}$
- key vectors are stacked into a key matrix $\mathbf{K}$
- $w$ can be written as dot product of $\mathbf{Q}$ and $\mathbf{K}$

# Transformer Models

## Attention Mechanism

- The weight matrix $w$ needs to be normalized
  - rescale the values by $d$ (dimensionality of the query and key vectors)
  - normalize the values using softmax to be non-negative and add up to 1
  - this yields the weights to calculate the updated token representations
- The attention mechanism combines the weights with the original embeddings
  - compute values $\mathbf{v}_i = \mathbf{W}_\mathbf{v}\mathbf{r}_i$, stacked into a value matrix $\mathbf{V}$, containing linear projections of the input embeddings
  - updated representations $\hat{\mathbf{r}}_i$ are summation of $\mathbf{V}$ weighted by $w$

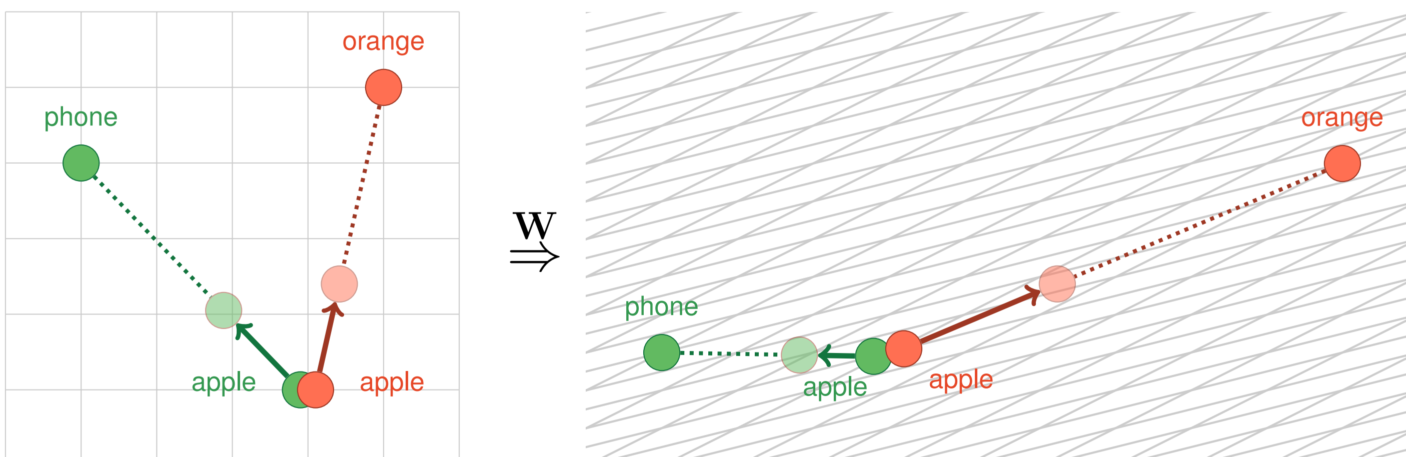
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \underbrace{\text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)}_{w} \underbrace{\mathbf{V}}_{\mathbf{W}_v\mathbf{R}^T}$$

- Learnable parameters are the weight matrices $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$ which encode the linear transformations of input embeddings $\mathbf{R}$

# Remarks

What does it mean to apply a linear transformation to the input embeddings and why do we do it?

- ❑ Linear transformation ➜ scale/shift of the input space given by a transformation matrix $\mathbf{W}$
- ❑ Consider two sentences ‘apple released their new phone’ and ‘an apple and an orange’
- ❑ Updated embeddings will move closer to their context words during the update step
  - – apple should move closer to phone (its input context is tech-related)
  - – apple should move closer to orange (its input context is fruit-related)
  - – both should move away from each other (increase their discriminative power)
- ❑ Optimum: the transformation matrix $\mathbf{W}$ that maximizes the information gained



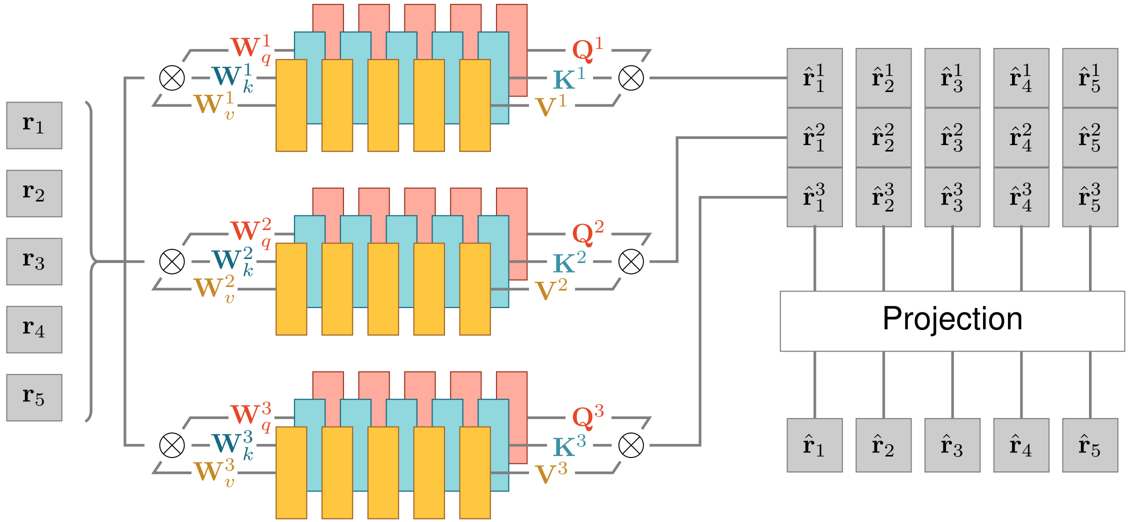

Since the optimal transformation is different for $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, each learns their own matrix $\mathbf{W}$.

# Transformer Models

## Multi-Head Attention

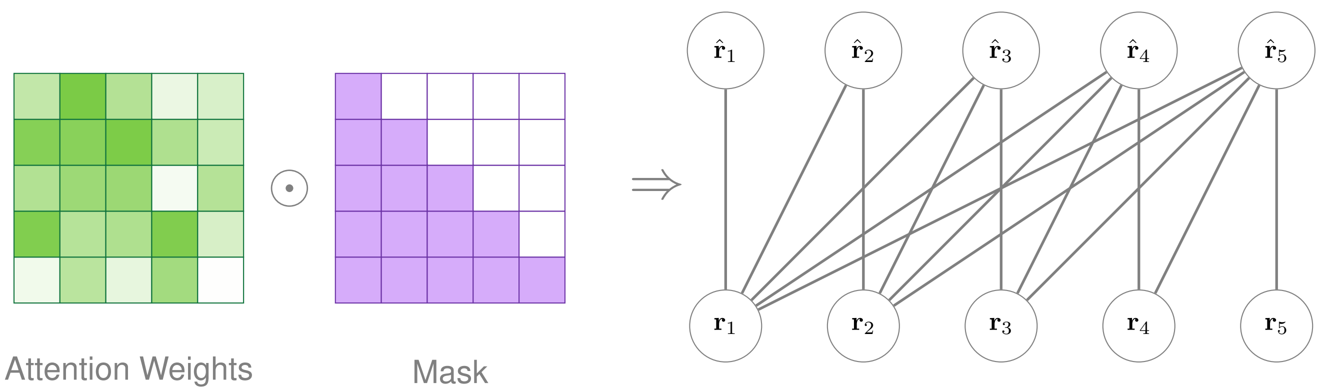
- Multiple attention mechanisms $1 \ldots l$ each with different learned parameters $\mathbf{W}_q^l$, $\mathbf{W}_k^l$,$\mathbf{W}_v^l$ called *heads* can be combined
- each head produces a different output; these are concatenated and passed through a projection to reduce their dimension back to the original
- as each attention head can learn different weightings of representations, they can encode different relationships between tokens in a sequence



# Transformer Models

## Masked Attention

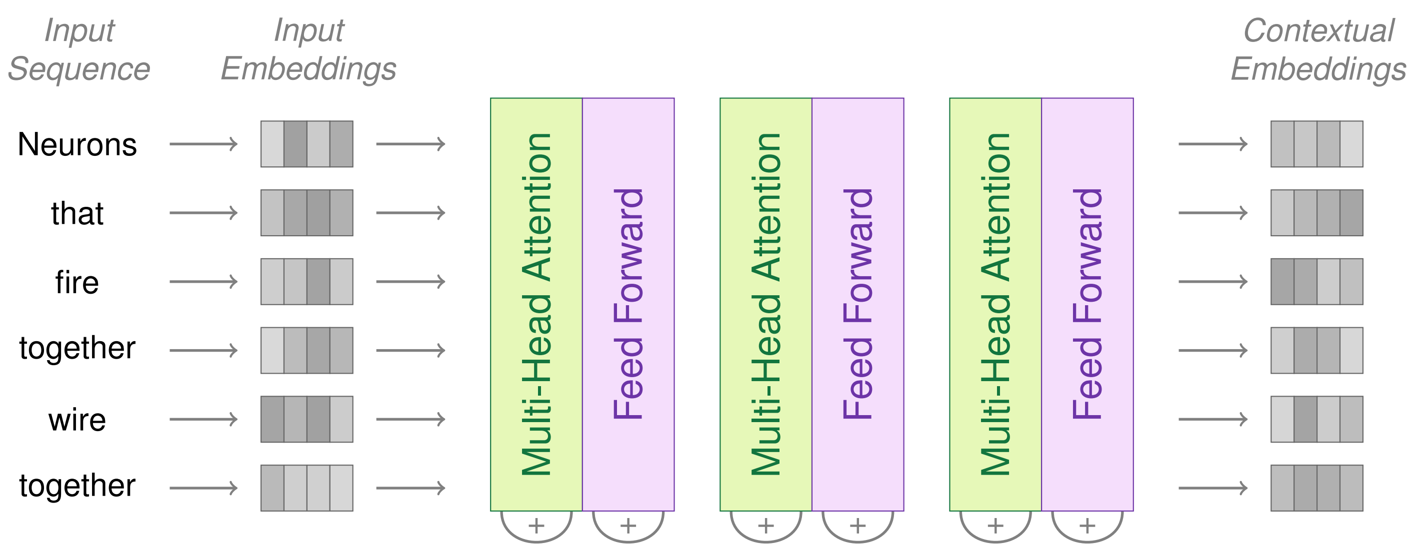
- Masking is used to prevent attention to certain tokens
- A binary mask is applied to the weight matrix
  - masking has to commence before normalization to not influence scores
  - masked scores are set to $-\infty$, thus resulting in a $0$ after the $\mathrm{softmax}$
- For example, the mask below can be used to have every token attend only to tokens before it (causal language modeling)



# Transformer Models

## Transformer Architecture

- Multiple attention blocks can be stacked to form a Transformer model
- A fully connected feed-forward layer is applied to each embedding separately and identically after each attention block
  - this allows the Transformer to learn complex relationships (non-linear)
  - repeated attention without would compute only weighted averages (linear)
- Residual connections ('+') are added to add a portion of the input back to the output of each layer (yields stabler gradients due to shorter signal path)



# Transformer Models

## Encoder-Decoder Models

- The stack from the previous slide is commonly called an Encoder
  ➜ produces contextualized embeddings for an input sequence
- It can be coupled with a Decoder which adds cross-attention
  ➜ applies encoder embeddings as query and keys to own values
  - decoder output is fed through linear layer predicting probabilities over the vocabulary
  - decoder receives causal mask and shifted input, restricting attention to previous tokens