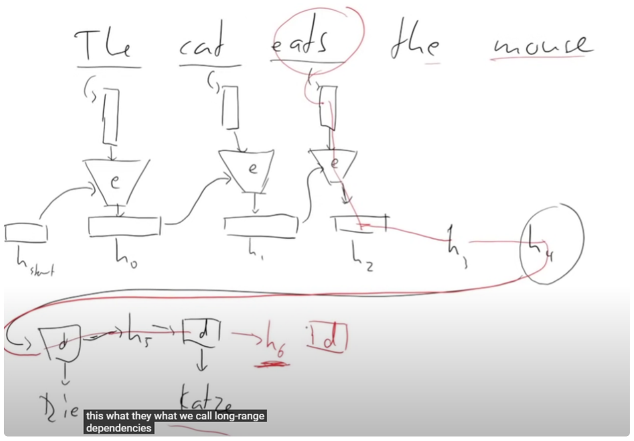
Transformer

The problem of RNN/LSTM: Long range dependency



To generate a translation of “eat”, have to remember all the hidden states(red line)

How attention could solve this problem?

Attention serve as filter (simliar as LSTM),

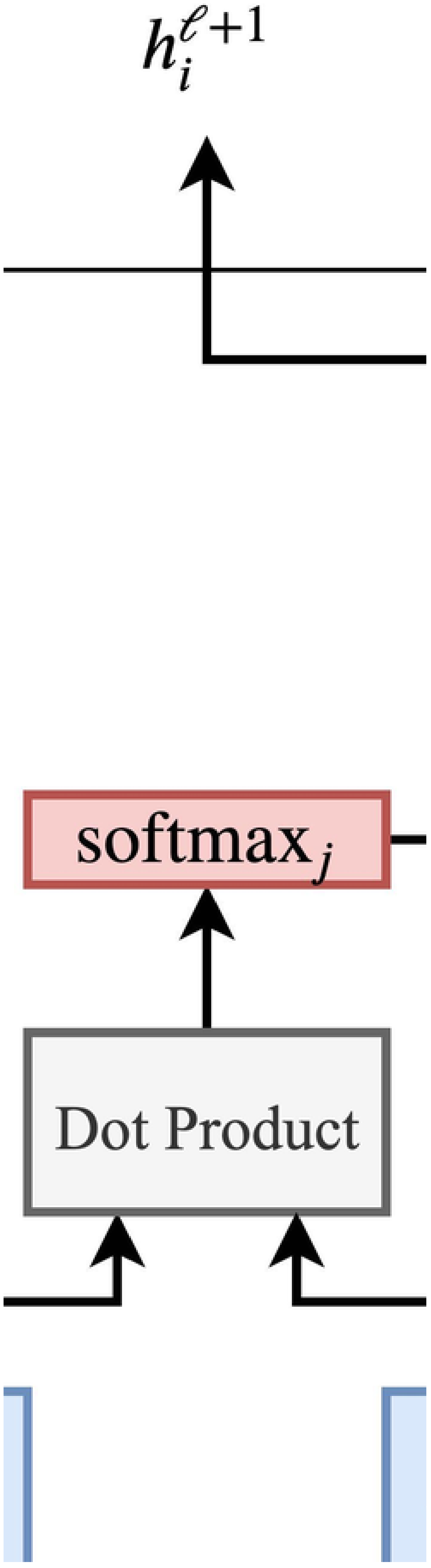
- source are embeded into two vectors - value (to store orignal value) & key (to help filter) (encoding)
- previous outputs are embedded into vector as query to help filter(encoding)
- calculate the dot product (similarity)for each query with all keys (decoding)
- softmax to select the most similar query-key pairs and filter out the others (decoding)

Note1. by filter out some hiden states, attention will help to solve long range dependency-- guess the reuslt will be more accurate and the NN will be simpler.

Note2, use previous prediction to filter input by **dot products in decoder,** whereas LSTM filter by the **weights in the encoder of input and all previous hidden state**

🍑 Attention? Attention!

Why Transformers are Graph Neural Networks?



ormer

tecture by transl

s and vectors. W

o layer $\ell + 1$ as

$$\text{tion}\left(Q^\ell h_i^\ell, K\right.$$

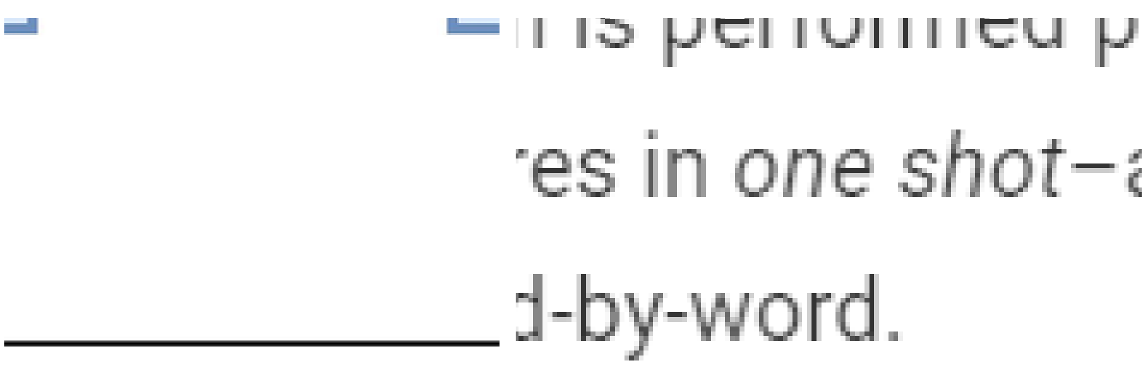
$$^{+1} = \sum_{j \in \mathcal{S}} w_{ij} \left(V\right.$$

$$\text{softmax}_j \left(Q^\ell h\right.$$

in the sentence

and Value for th

is performed n



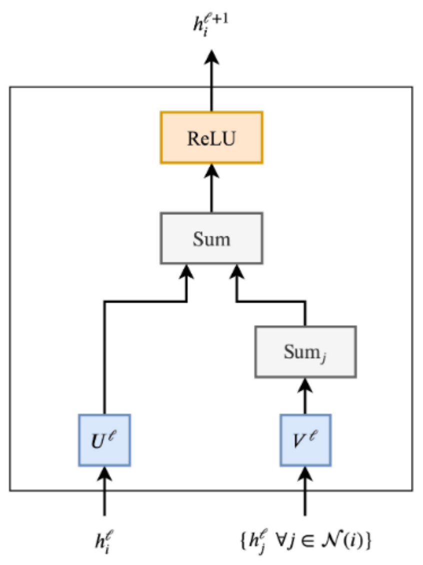
is performed p

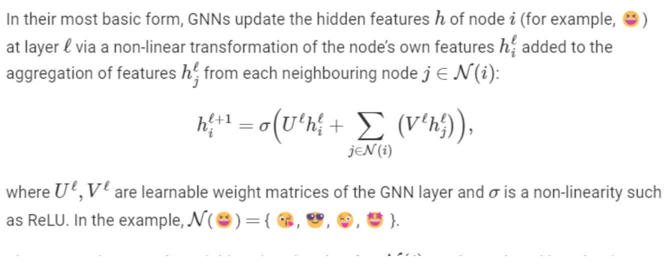
es in one shot–a

d-by-word.

Transformer:

update h_l+1: for current query i (the work you are going to translate/predict), calculate similiaty of it with all input embeded as key_j as filter/(another cuz QKV are embedding vectors)weight, after selection/application, **h_l+1 = sum over all weighted input value_j (including i itself input)**





In their most basic form, GNNs update the hidden features h of node i (for example, 😄) at layer ℓ via a non-linear transformation of the node's own features h_i^ℓ added to the aggregation of features h_j^ℓ from each neighbouring node $j \in \mathcal{N}(i)$:

$$h_i^{\ell+1} = \sigma\Big(U^\ell h_i^\ell + \sum_{j \in \mathcal{N}(i)} \left(V^\ell h_j^\ell\right)\Big),$$

where U^ℓ, V^ℓ are learnable weight matrices of the GNN layer and σ is a non-linearity such as ReLU. In the example, $\mathcal{N}$(😄) = { 😘, 😎, 😜, 😍 }.

GNN:

update h_l+1: for the target node i, embedding all its neighbors with weight, **h_l+1 = sum over weighted i and all its neighbors**

Difference:

transformer softmax first (filter) then sumover

GNN sum first then Relu

Click here to expand...

If we were to do multiple parallel heads of neighbourhood aggregation and replace summation over the neighbours � with the attention mechanism, *i.e.*, a weighted sum, we'd get the **Graph Attention Network** (GAT). Add normalization and the feed-forward MLP, and voila, we have a **Graph Transformer**!