LING3804: Lecture Notes 7
Similarity and Attention in Transformer Models

This lecture shows how neuronal activation defines similarity, which is used to model attention.

Unlike backpropagation, attention in models like transformers stays effective at large scales.

Contents

7.1 Similarity in neural activation

Recall from the lecture on associative memory how to cue (multiply) a matrix by a vector:

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \overset{\text{def}}{=} \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Let's look at each single neuron:

$$p_1 \overset{\text{def}}{=} \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Activation p_i is derived from the product of weights $w_{i,j}$ and antecedent neuron activation a_j:

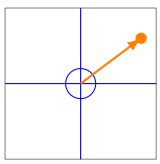
$$p_i \overset{\text{def}}{=} f\left(\sum_j w_{i,j}\, a_j\right)$$

If metabolism keeps weight and activation patterns at unit magnitude (sum of squared elements = 1)

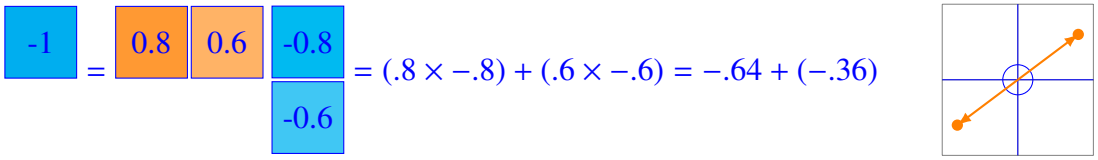
(and if we assume a linear threshold function $f(x) = x$ or, equivalently, ignore it):

1. the resulting activation is maximized at 1 when the vectors match:

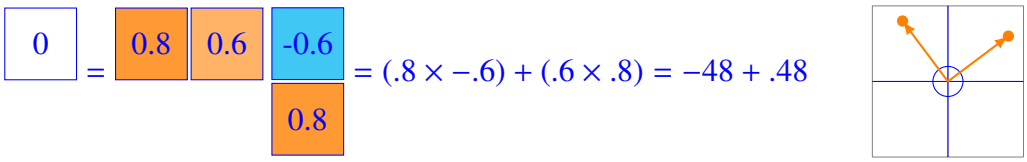
$$\begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} 0.8 & 0.6 \end{bmatrix} \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix} = (.8 \times .8) + (.6 \times .6) = .64 + .36$$



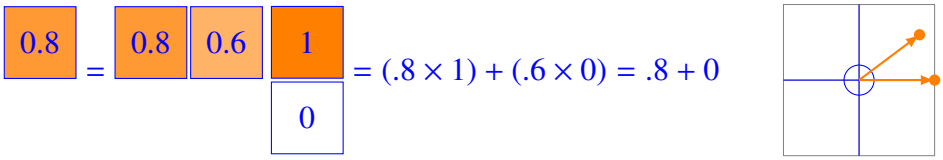
2. the resulting activation is minimized at -1 when the vectors oppose:



3. the resulting activation is 0 when the vectors are orthogonal (at right angles; independent):



4. the resulting activation varies smoothly between these outcomes:

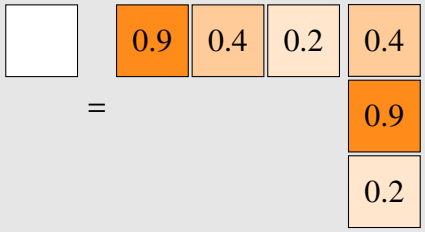


This forms a **similarity** measure!

Specifically, it's the **cosine** of the angle between vectors (length of adjacent edge over hypotenuse).

Practice 7.1:

What is the similarity between the below vectors:



7.2 Attention [Vaswani et al., 2017]

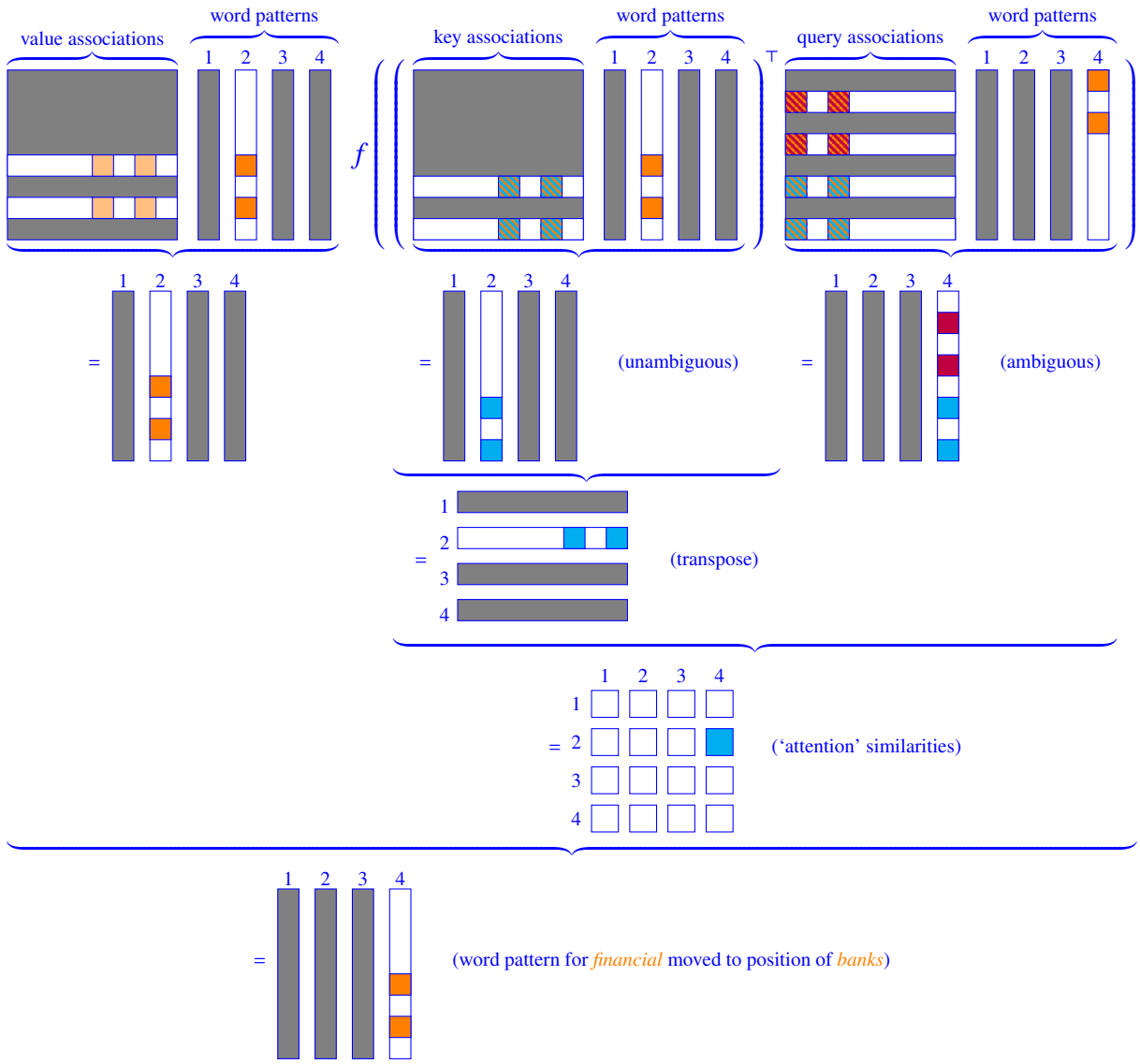
Transformer language models use this similarity to find words to pay attention to when predicting.

Here's how they do it:

1. look up **query/key/value** pattern for each word, using learned association from word pattern
2. compute 'attention' similarity for **query** and **key** pattern of each current and previous word

3. weight **value** pattern for each previous word by this attention similarity to each current word

For example, in processing *In financial markets, banks…*, *banks* could mean *river-* or *money-*:



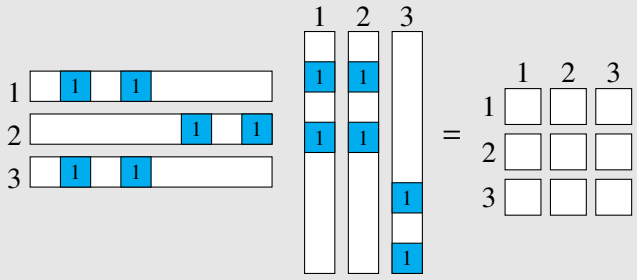
The resulting pattern at *banks* looks like the word pattern for *financial*, disambiguating the sense!

Result patterns are then added to the word patterns and fed into another layer to repeat the process.

Transformers have dozens of layers, all repeated for dozens of 'heads', allowing manifold attention.

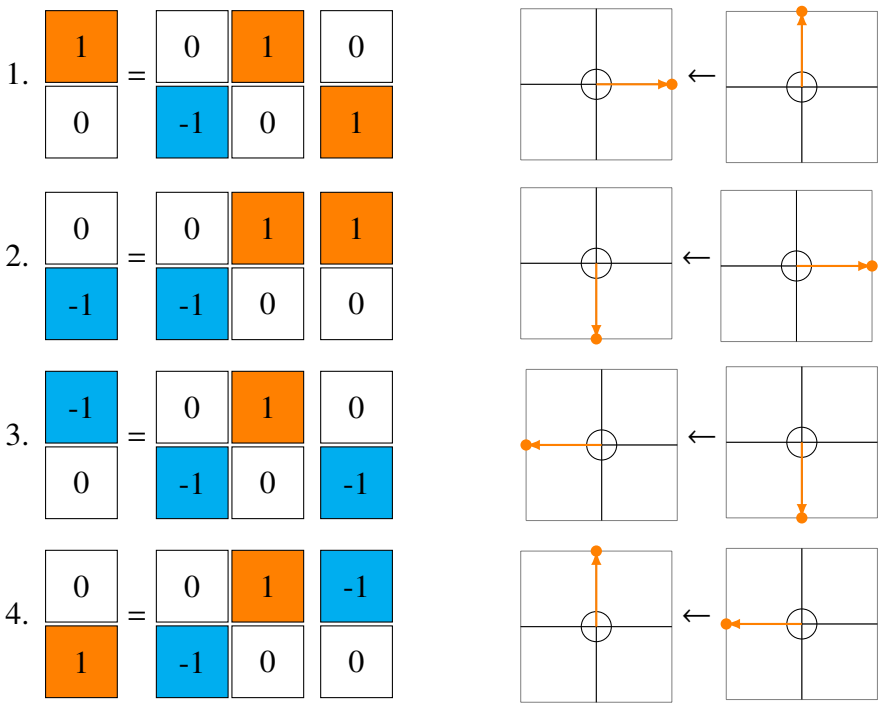
Practice 7.2:

Fill in the similarities in the resulting attention matrix:



7.3 Positional encoding [Vaswani et al., 2017]

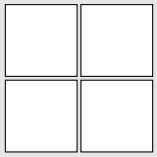
Note that a simple one-layer network can encode fixed-degree rotations.



We can use these rotations to encode position offsets.

Practice 7.3:

Write a matrix to encode a backward single-position offset using the above position vectors:



7.4 In-context learning [Olsson et al., 2022]

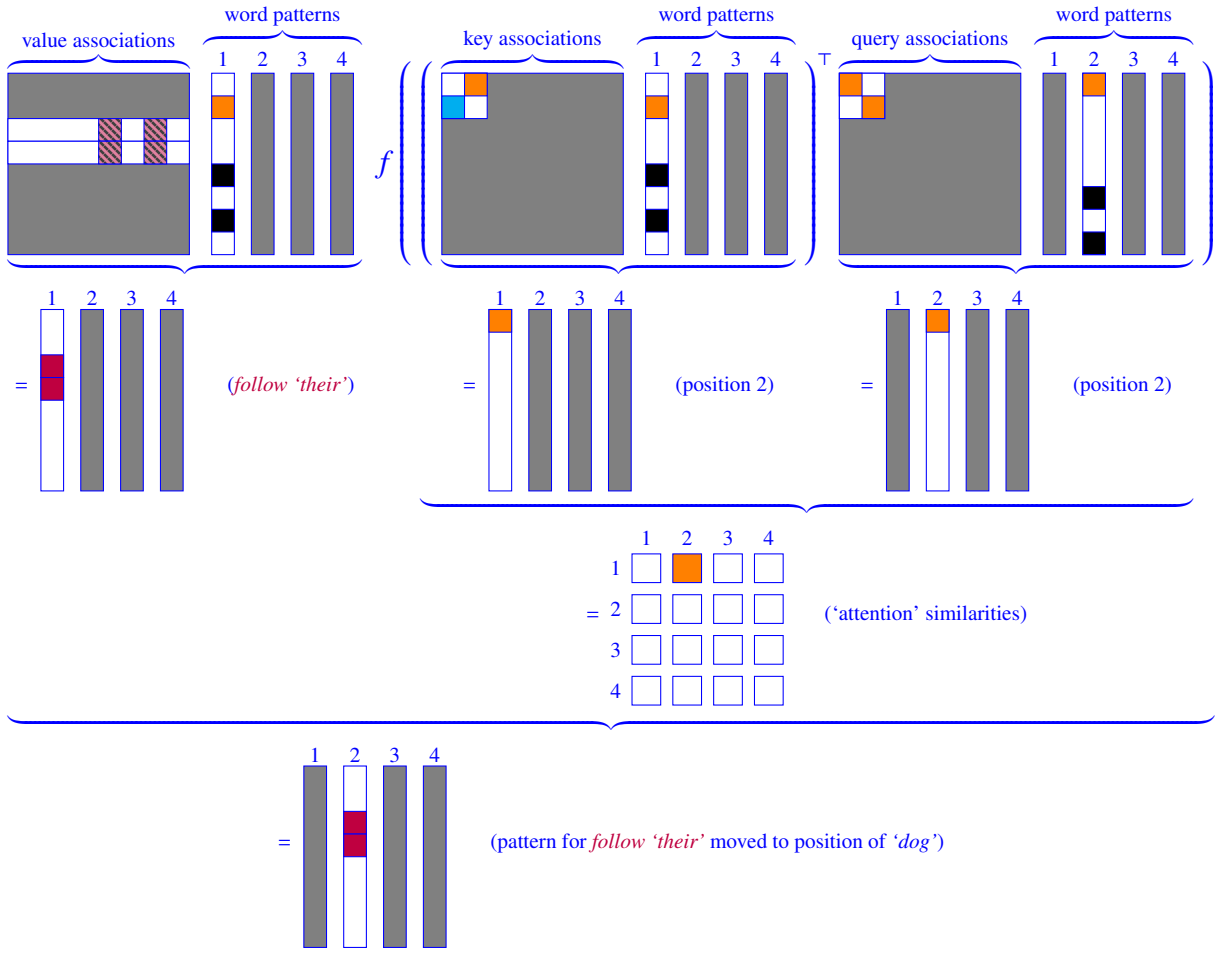
Transformers can learn to repeat content from its input without having seen that content before!

One way this happens is with an 'induction head': e.g. we may predict *dog* after *their dog . . . their*.

First, at some layer, the system learns to direct its attention to an offset from an input word *their*

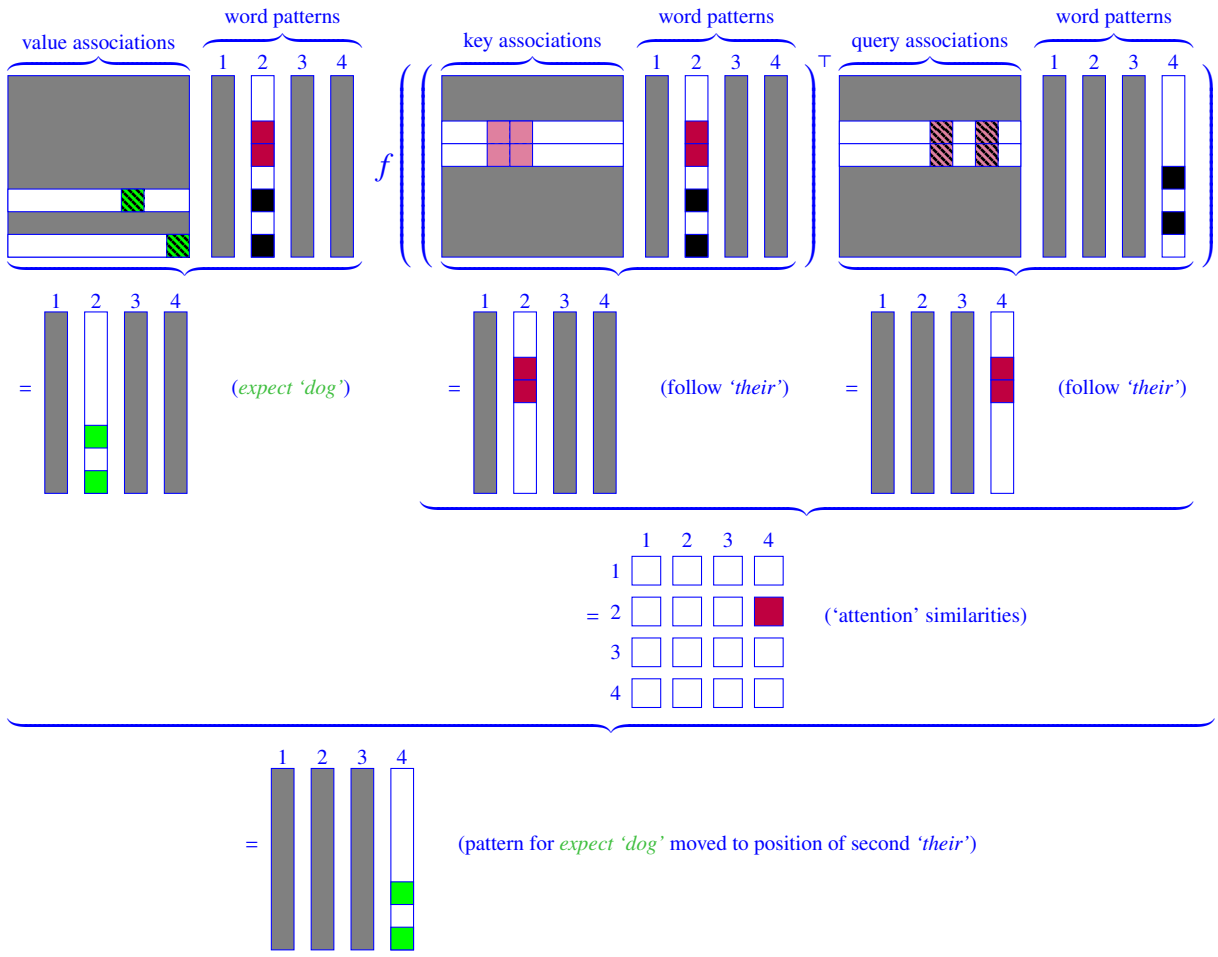
(this offset is implemented using the first two elements of each vector as a positional encoding)

(the **key** matrix for some attention head offsets one position and the **query** matrix offsets zero):



These resulting vectors are sent on to the next higher layer, where the word vector gets moved.

At the next layer, the model expects the word at that offset from one or more previous instances:



(Assume for simplicity that the resulting pattern here is directly used to predict the next word.)

Note that ***no*** pre-trained **query/key/value** matrix associates *follow 'their'* with *expect 'dog'* — and if the value matrix is identity (diagonal ones), the expected word ***need not*** be seen in training!

This is called **in-context** learning: it happens in the context and doesn't require training examples. Heads that behave this way are relatively simple and arise spontaneously during model training.

References

[Olsson et al., 2022] Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., & Olah, C. (2022). In-context learning and induction heads.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In *NIPS* (pp. 5998–6008).