**Attention mechanisms for text generation**

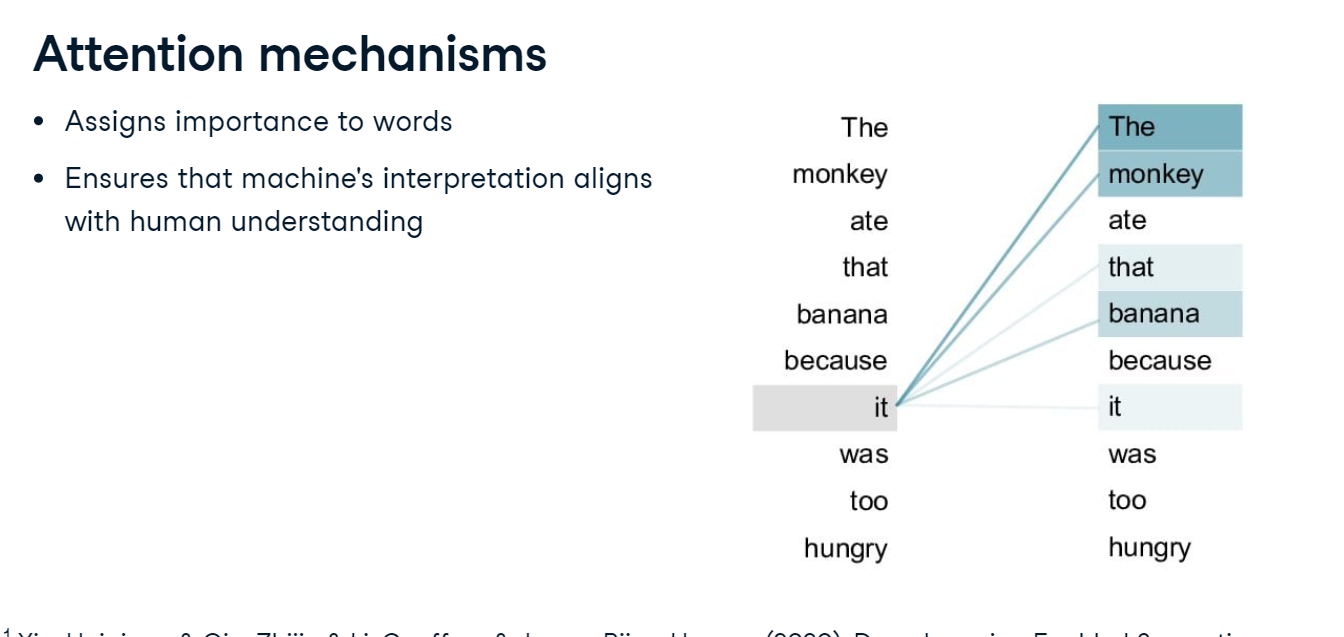
Let's review the transformer's attention mechanism.

**The ambiguity in text processing**

Look at this sentence: The monkey ate that banana because it was too hungry. For humans, pinpointing what the word "it" refers to is an easy task. The ambiguity of "it" can lead to misunderstandings in machines, especially in tasks like translation.

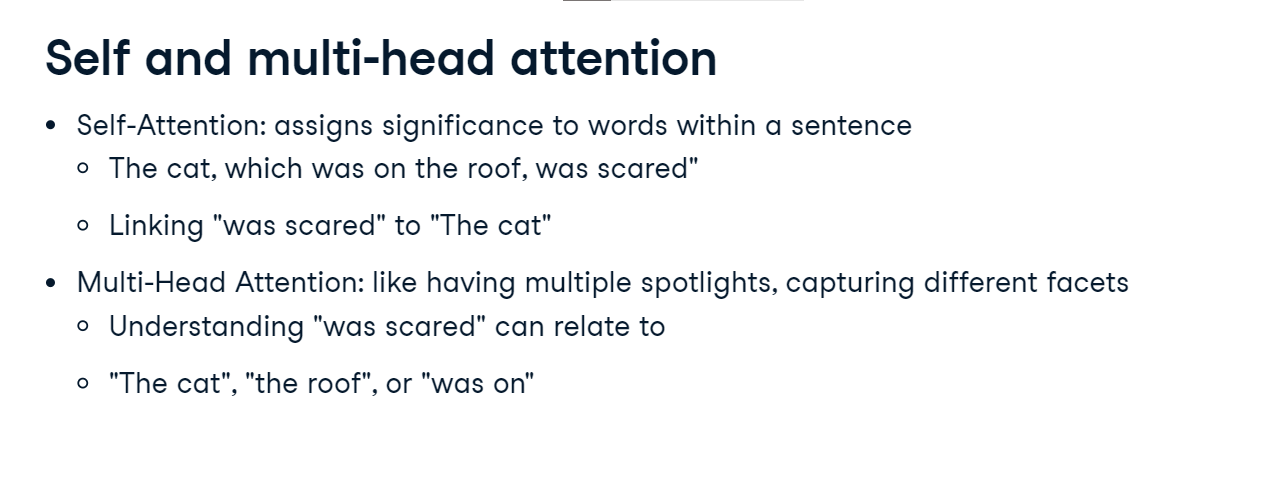
**Attention mechanisms**

The Attention Mechanism in the transformer assigns importance to words within a sentence. For our example, 'it' is understood to be more related to 'the','monkey' and 'banana' in descending order of significance. This ensures that in tasks like translation, the machine's interpretation aligns with the human understanding.

****

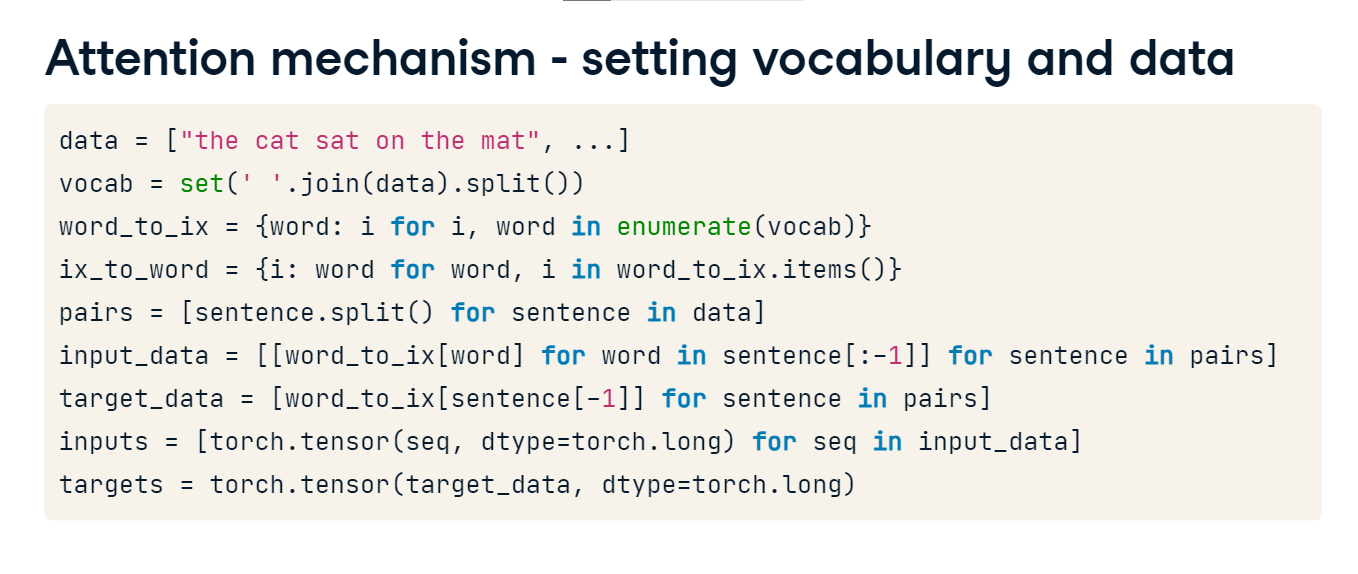
**Self and multi-head attention**

The attention mechanism contains both self and multi-head attention. Self-Attention assigns significance to words within a sentence. In "The cat, which was on the roof, was scared," the mechanism links "was scared" directly to "The cat". Multi-Head Attention is akin to deploying multiple spotlights. In the same example, "was scared" could relate to "The cat," signify "the roof," or point to "was on".



**Attention mechanism - setting vocabulary and data**

Let's explore a self-attention example with a synthetic dataset named data. Using this, we create a vocabulary set. For efficient processing, we map each word to an index for training and vice versa for testing. Sentences are transformed into input and target pairs, where input contains all words except the last, and the target is the last word.



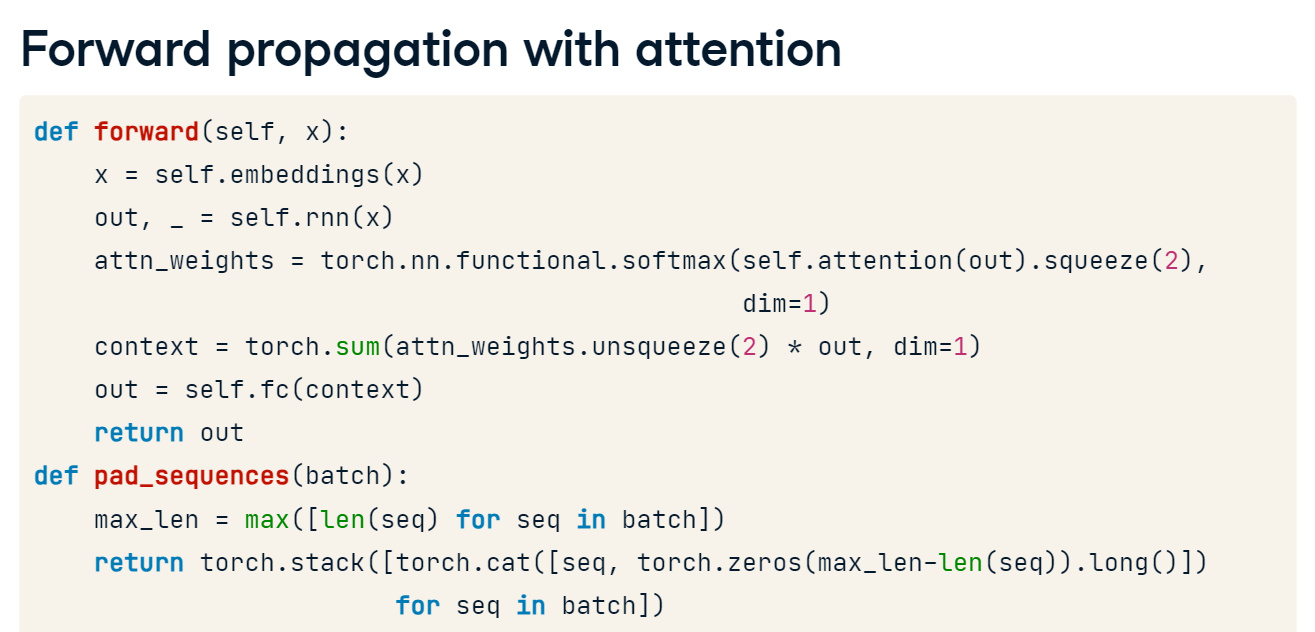
**Model definition**

We set dimensions with an embedding\_dim of 10 and hidden\_dim of 16, balancing granularity and efficiency; increasing them may lead to overfitting. RNNWithAttentionModel extends nn-dot-Module; it houses an embedding layer translating word indexes to vectors and an RNN layer for sequential processing. The attention layer computes word significance scores, performing a linear transformation of hidden\_dim to one, yielding a singular attention score per word. Finally, the fc layer, outputting vocab\_size, pinpoints the predicted word's index.



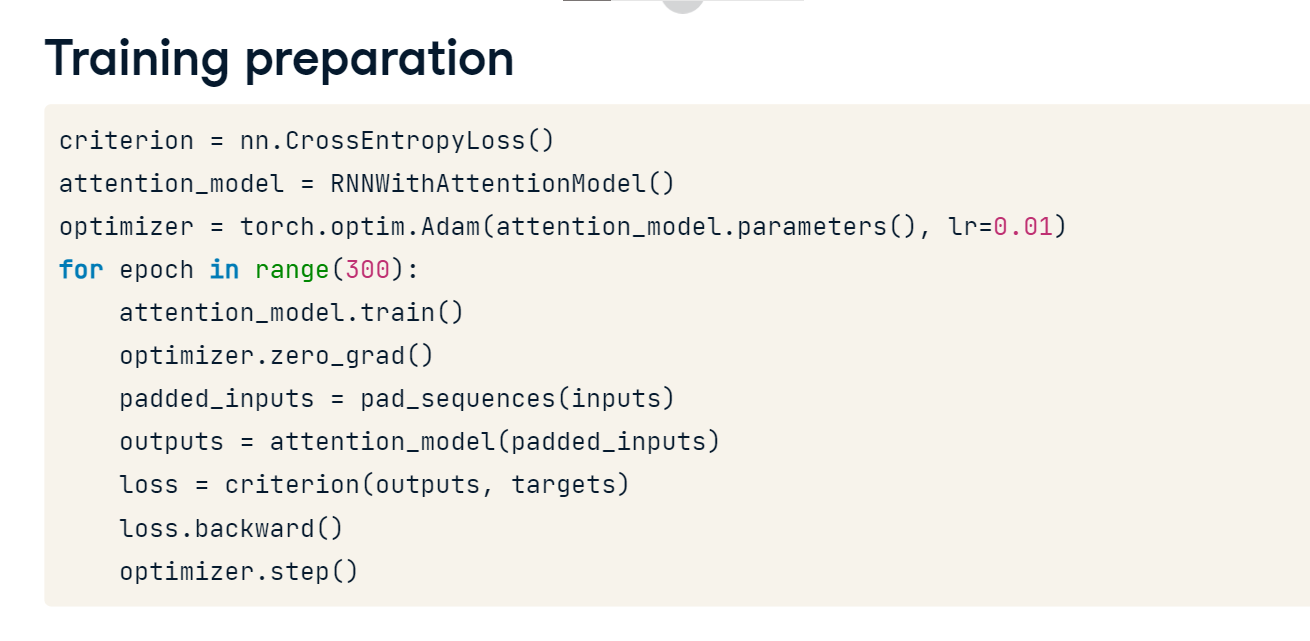
**Forward propagation with attention**

In the forward method, word indexes are embedded, and the RNN layer processes them, generating outputs for each word. Next, attention scores are derived by applying a linear transformation to the RNN outputs, normalizing using softmax, and reshaping the tensor using squeeze two to simplify attention calculations. Following, a context vector is formulated by multiplying attention scores with RNN outputs, creating a weighted sum of the outputs, where weights are the attention scores. The unsqueeze two operation is important for adjusting tensor dimensions for matrix multiplication with RNN outputs. The context vector is then summed using torch-dot-sum to feed into the fc layer for the final prediction. Lastly, the pad\_sequences function ensures consistent sequence lengths by padding the input sequences with torch-dot-cat and torch-dot-stack, avoiding any potential length discrepancies and errors.



**Training preparation**

We initialize criterion with CrossEntropyLoss. After initializing our model, we choose the Adam optimizer for adaptive gradient updates. The training loop spans 300 epochs, with the model entering training mode at each epoch's beginning, and gradients reset to avoid accumulation. The pad\_sequences function ensures consistent input dimensions, after which the model generates predictions. The criterion measures how off these predictions are, and via backpropagation, updates the model's weights.



**Model evaluation**

We begin evaluation by converting input sequences into tensors with an added batch dimension using unsqueeze, allowing batched processing. Transitioning the model to evaluation mode ensures deterministic behavior. The highest score in the model's output corresponds to the predicted word, and torch-dot-argmax retrieves this index. With this index, we look up our vocabulary and present the input, the true target, and our model's prediction, showcasing its efficacy in context prediction. The output shows the next word in the sequence was correctly predicted.

