Lecture Notes for

Machine Learning in Python

⟦👨‍🏫, 👨‍💻, 🐍, 👨‍🔬⟧

Professor Eric Larson

Sequential CNN and Transformers

Lecture Agenda

- Logistics
  - Grading Update
  - Lectures
  - Sequential Networks due **during finals**
- Agenda
  - CNNs for Sequential Processing (review)
  - Transformers

Class Overview, by topic

Table Data Visualization

Numpy, Pandas, Seaborn
Overviews with some in-depth discussion

Dimension Reduction and Image Processing

Scikit-learn, Scikit Image,
Intuition only, Some mathematics

Linear and Logistic Regression

Numpy, Recreate API for Scikit-learn
Detailed mathematics for simple optimization
intuition for advanced optimization

Neural Networks and Back Prop.

Numpy
Detailed mathematics for NN operations

Wide and Deep Networks

Convolutional Networks

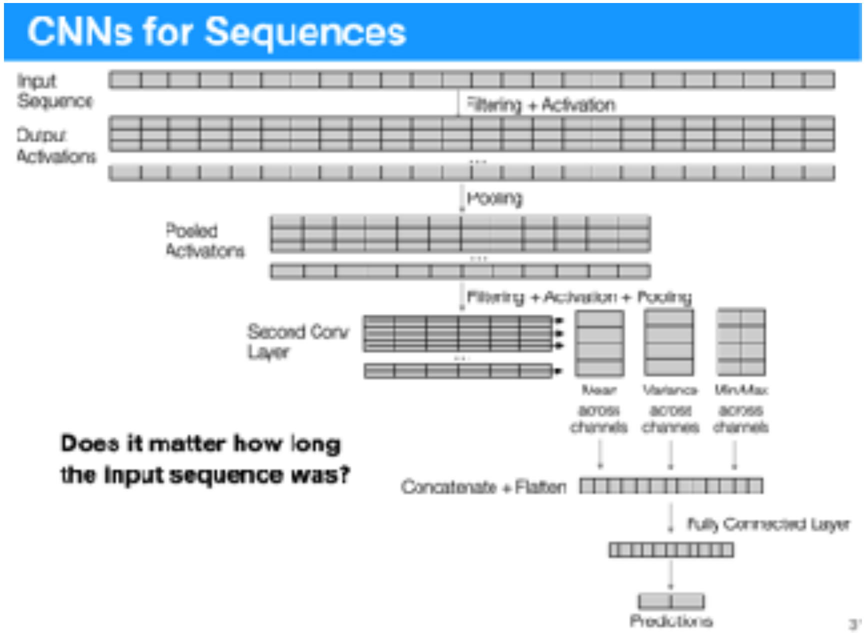
Sequential Networks

Keras, Tensorflow
Intuition, Detailed implement.

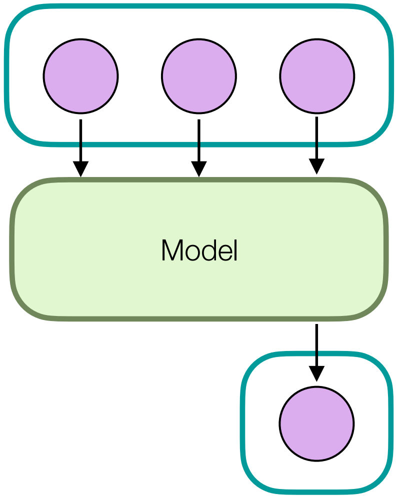
Ethics in Language Models

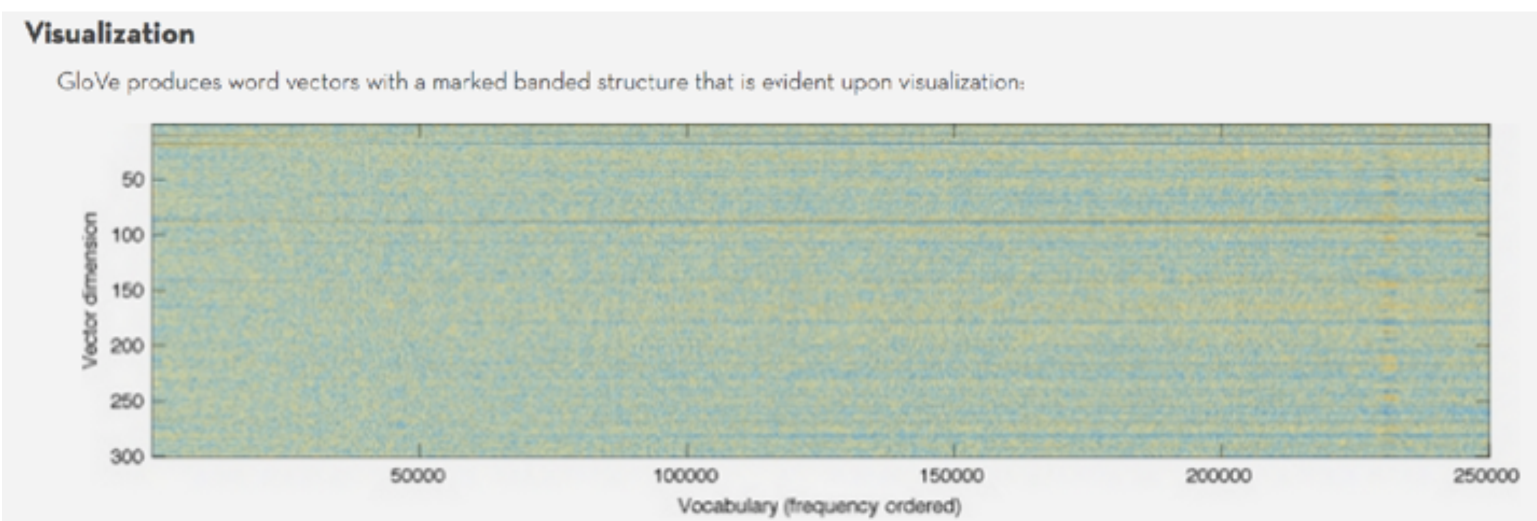
ConceptNet
Case studies

Last Time:



Many to One

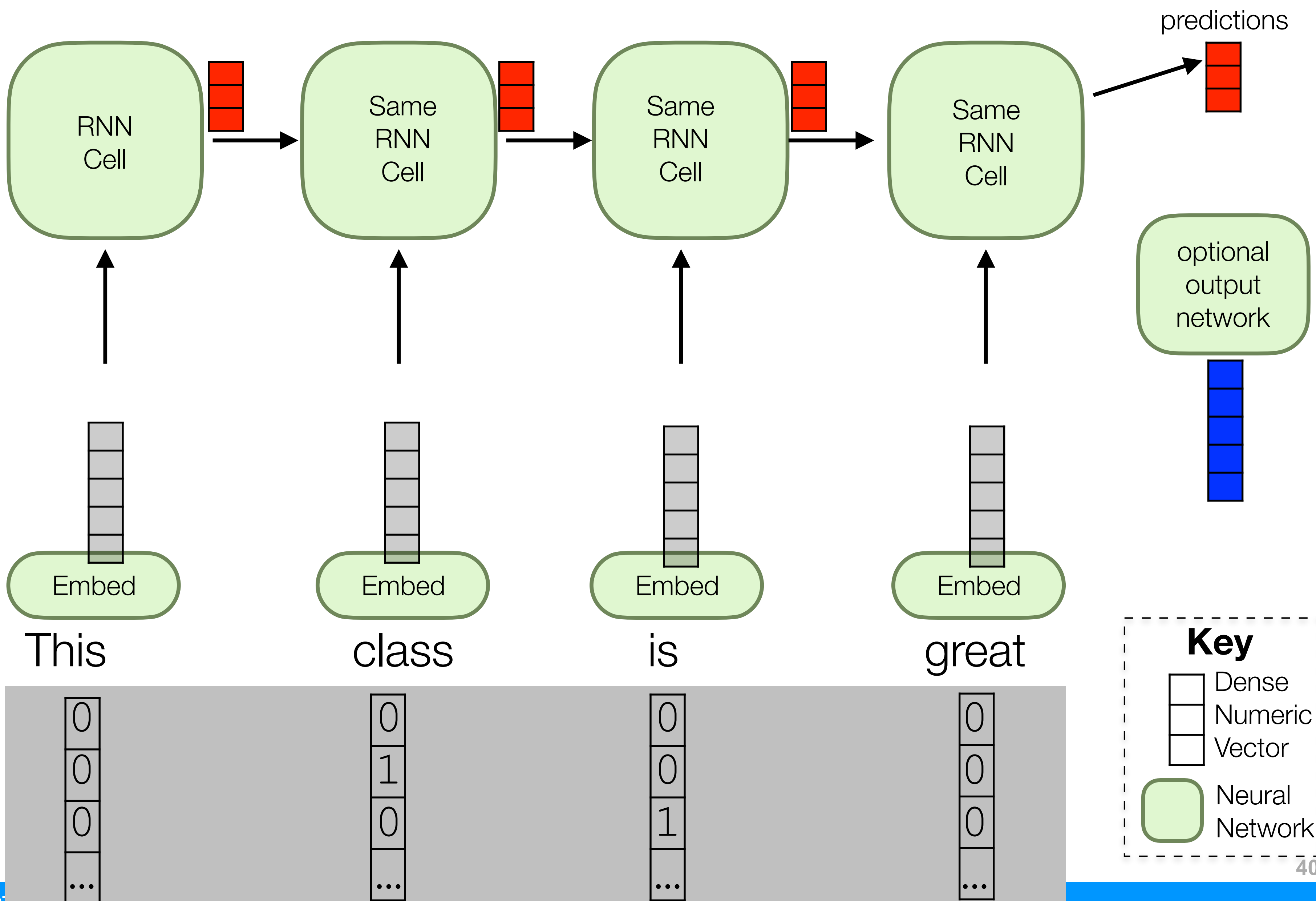




RNNs for Sequences

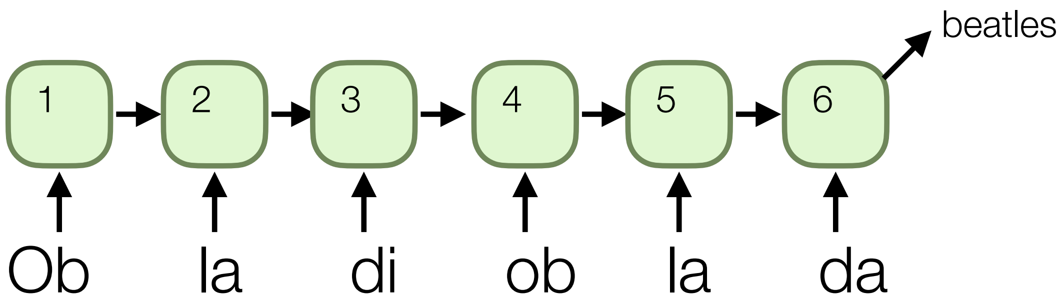
"And how does it make you feel when she jumps over you and calls you a lazy dog?"

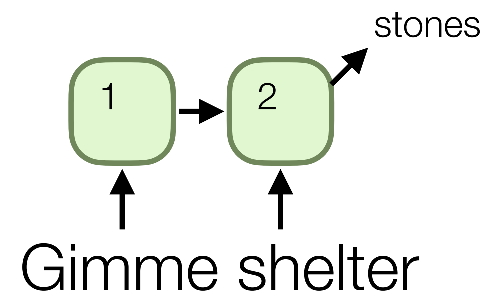
Recurrent flow with embeddings



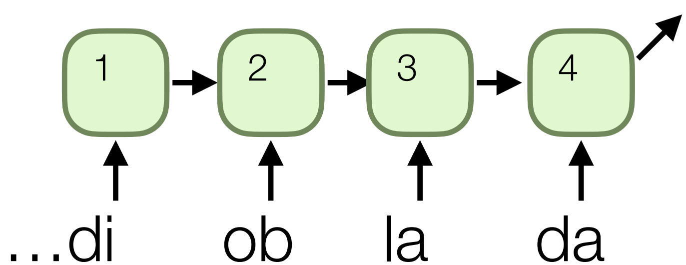
Different length input documents?

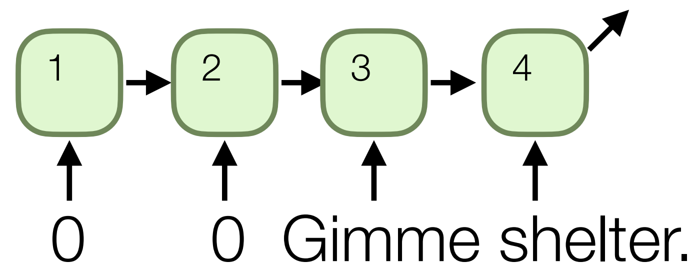
- option A: dynamic length sequences





- option B: padding/clipping





- main difference:

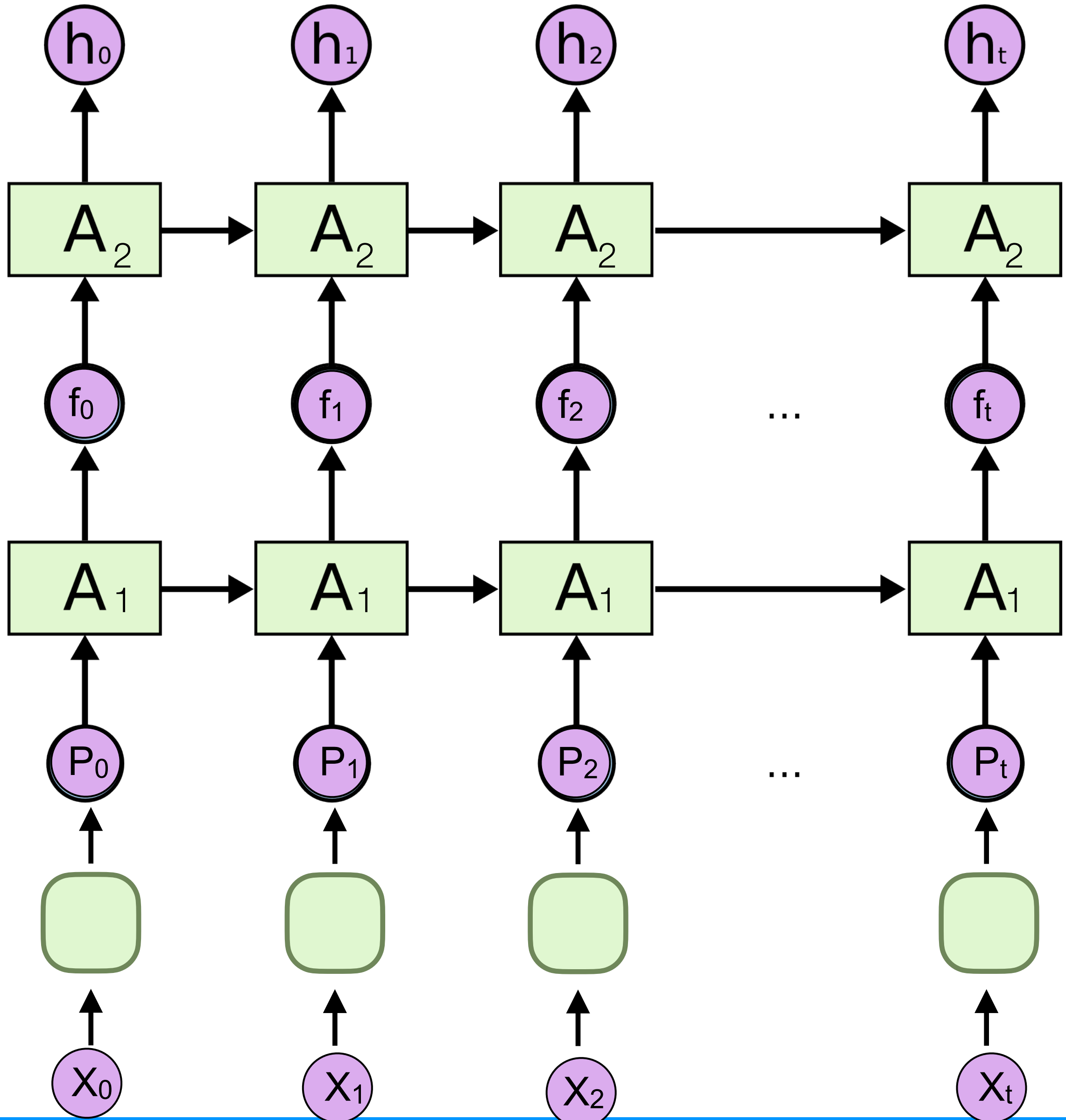
  speed based on computation graph design

Self Test

- The main reason **dynamic length is slow** is because:
  - A. the computation graph must be updated
  - B. weights must be tied together for each recurrent/sequential node differently
  - C. the embedding matrix cannot be applied in parallel to each word
  - D. no reason: dynamic length is actually faster

Sequence Stacking

keras:
return sequences

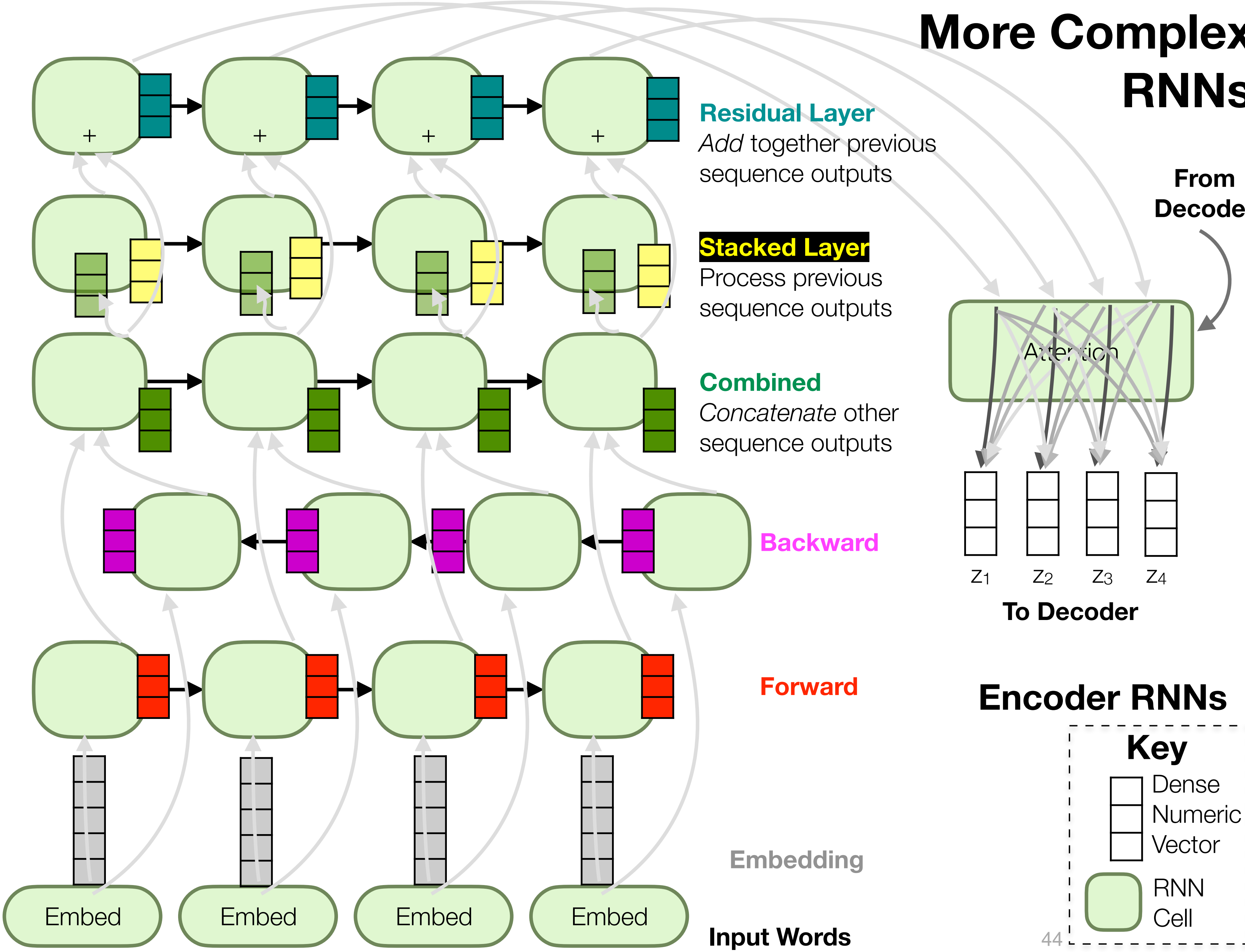


keras:
add final dense layer for prediction from last element only

keras RNN default:
return last element only
go to dense layer for prediction

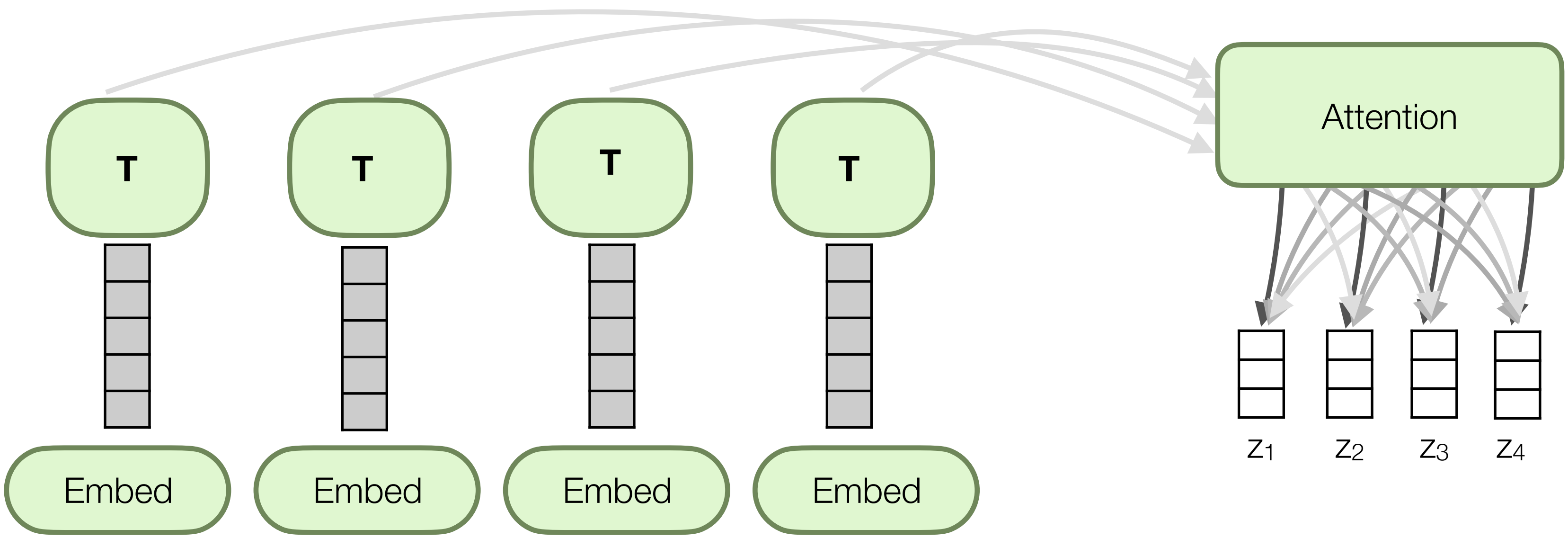
keras embed:
length of embedding is meant for sequences

More Complex RNNs



Transformers Intuition (reminder)

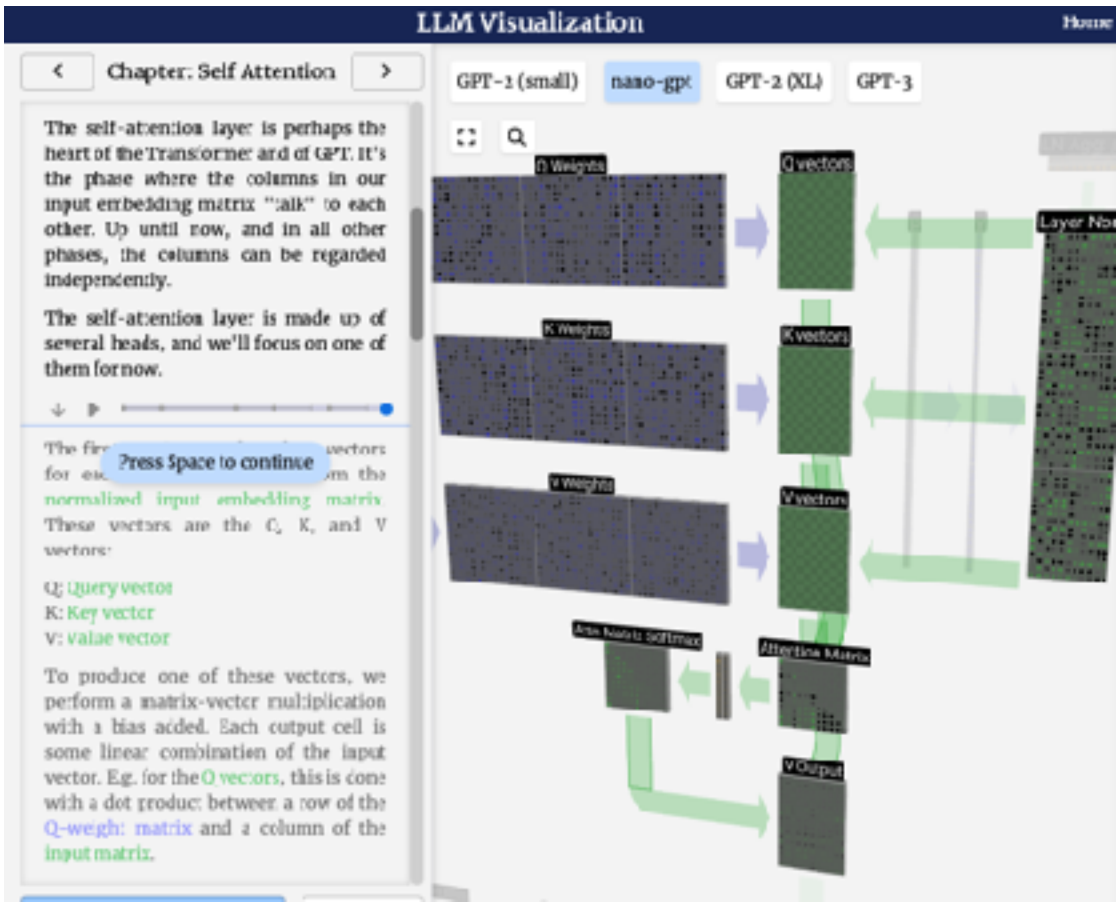
- Recurrent networks track state using an “updatable” state vector, but this takes processing iterative
- Attention mechanism (in RNNs) already takes a weighted sum of state vectors to generate new token in a decoder
- … so why not just use attention on a transformation of the embedding vectors? **Do away with the recurrent state vector all together?**



This link is perhaps the greatest tutorial on X-formers I have ever seen

Transformers

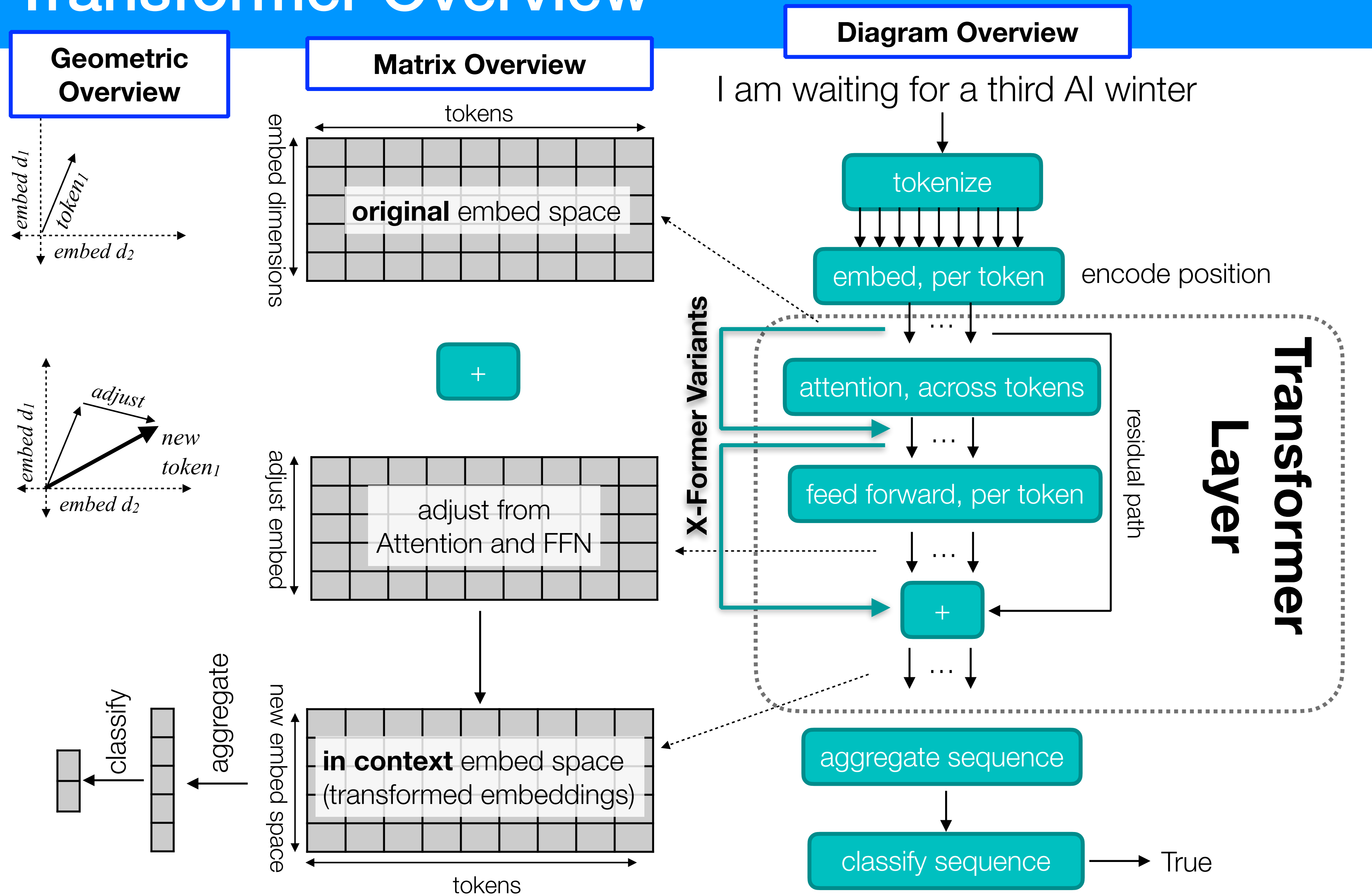
https://bbycroft.net/llm



Attention is All You Need

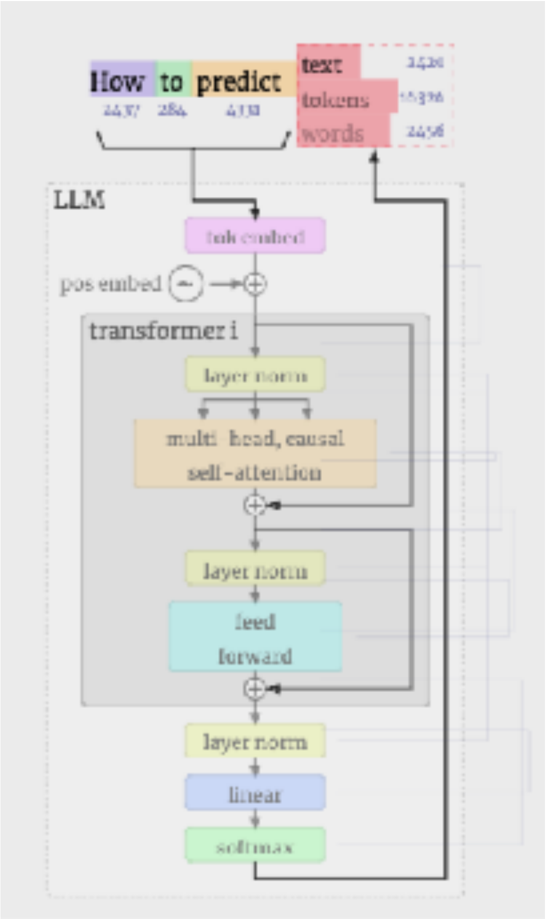
- **Continued Motivation**:
  - RNNs are not inherently parallelized or efficient at remembering based on state vector
  - CNNs are not resilient to long-term word relationships, limited by filter size
- **Transformer Solution:**
  - Build attention into model from the **beginning**
  - Compare all words to each other through **self-attention**
  - Only update **existing embedding space,** via residual
  - Define a notion of "**position**"in the sequence
  - ***Should be resilient to long term relationships and be highly parallelized for GPU computing!!***

Transformer Overview



Auto Regressive Transformer

https://bbycroft.net/llm



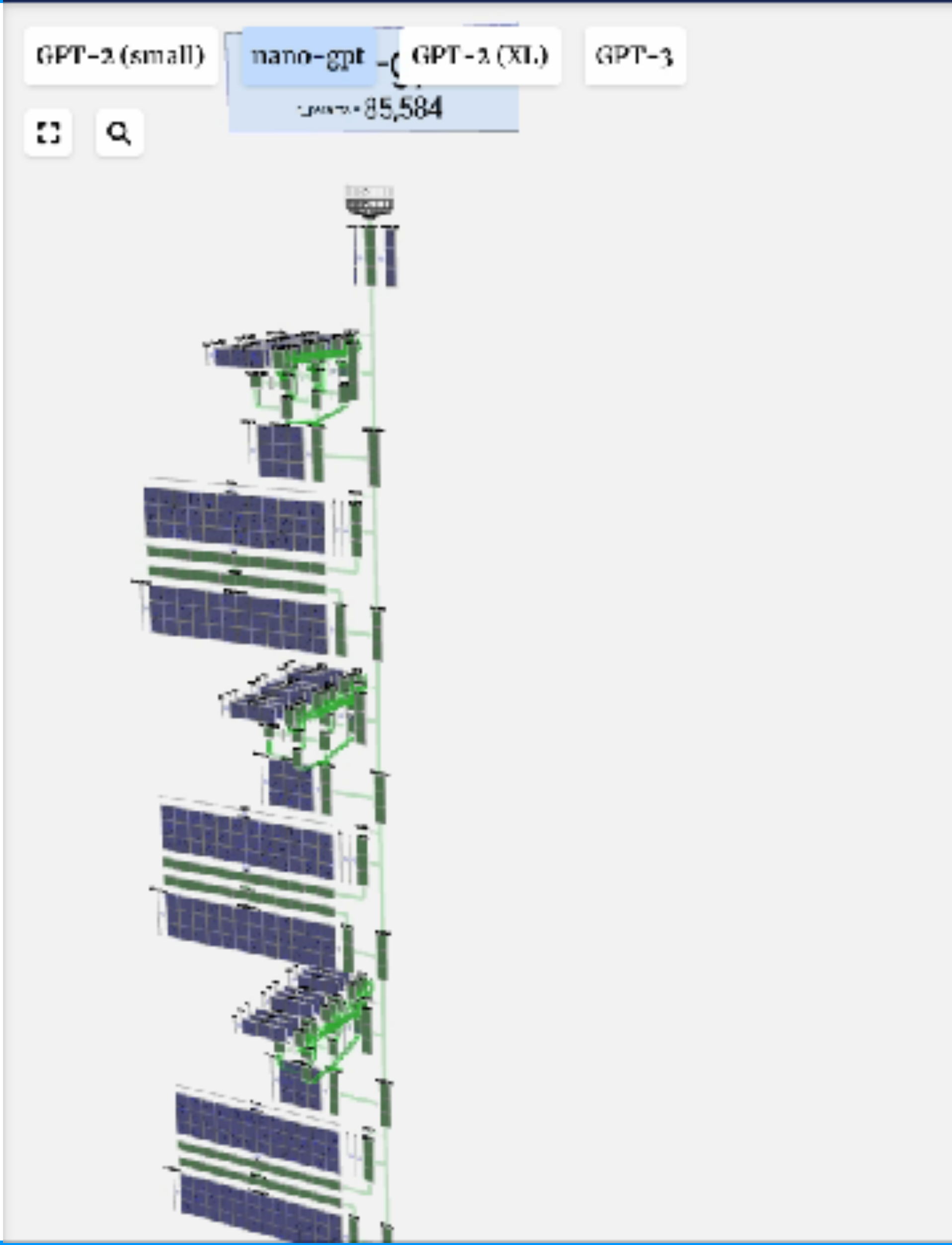
Components

- Embedding
- Layer Norm
- Self Attention
- Projection
- MLP
- Transformer
- Softmax
- Output

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

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

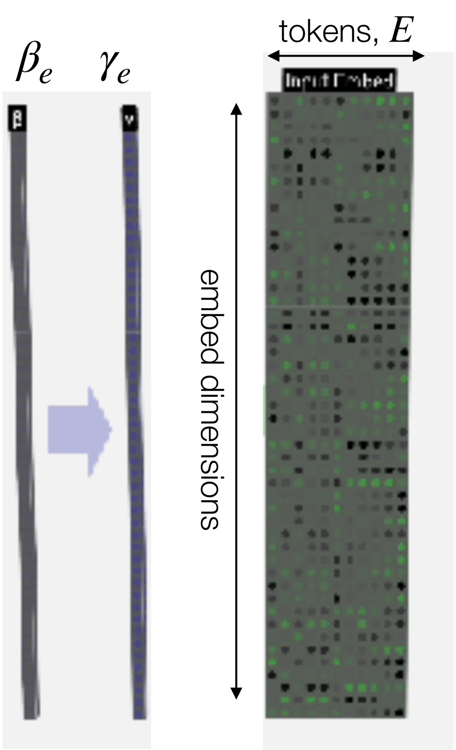


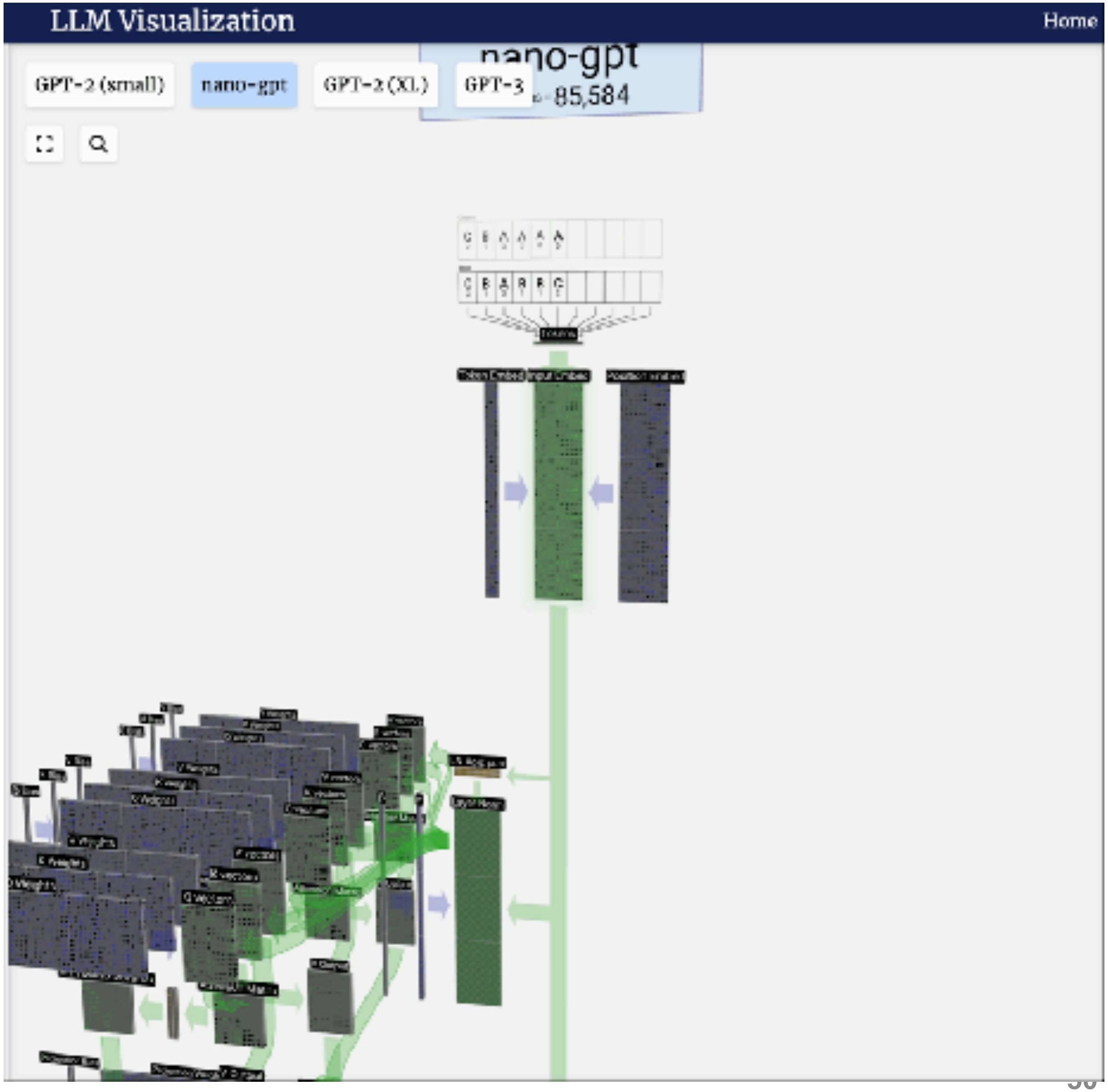
Layer Norm

$$LN(E_{tok}) = \gamma_e \left(\frac{E_{tok} - \mu_{tok}}{\sqrt{\sigma_{tok}^2 + \epsilon}} \right) + \beta_e$$

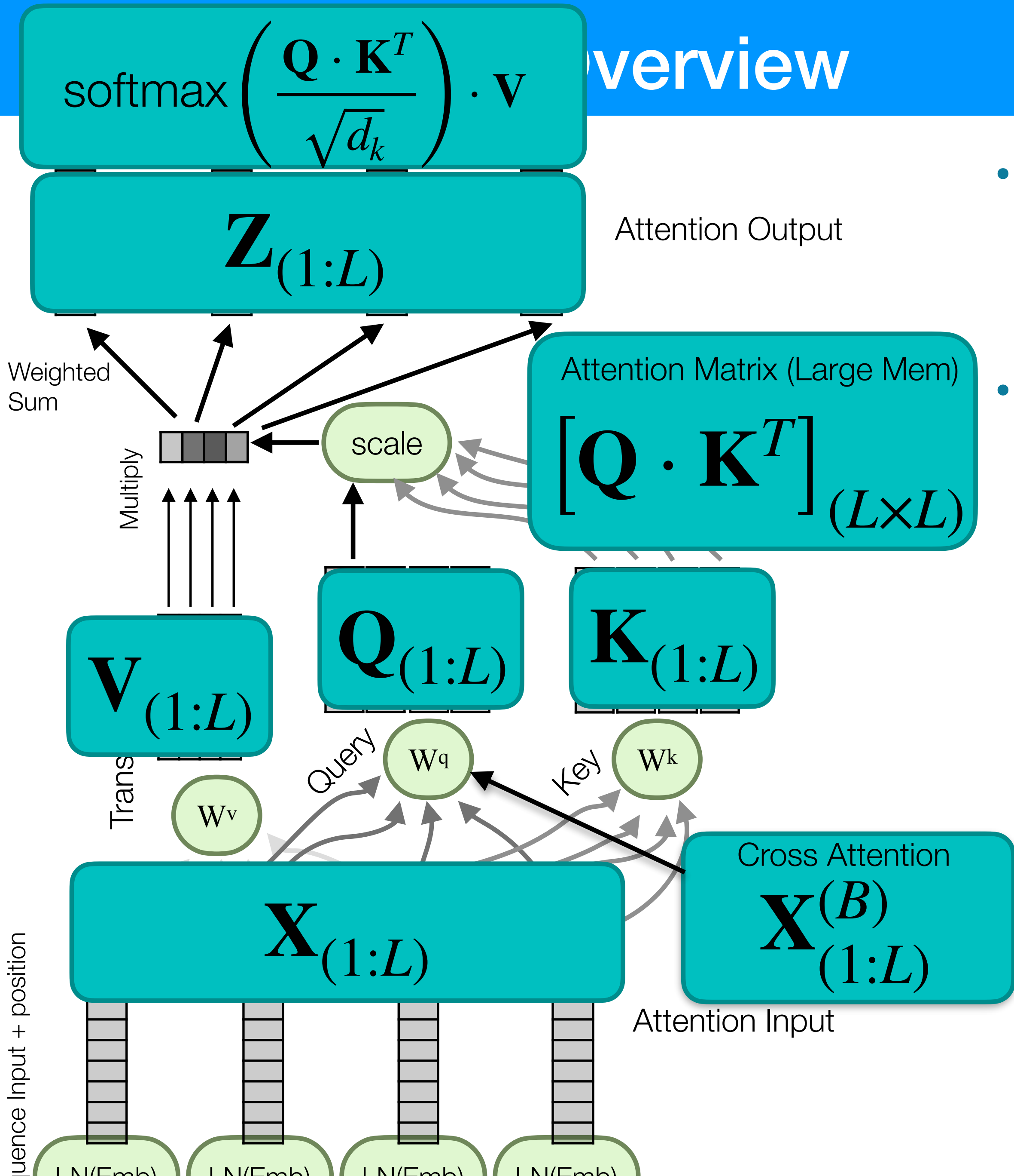
z-score per token

adjust each embed dimension



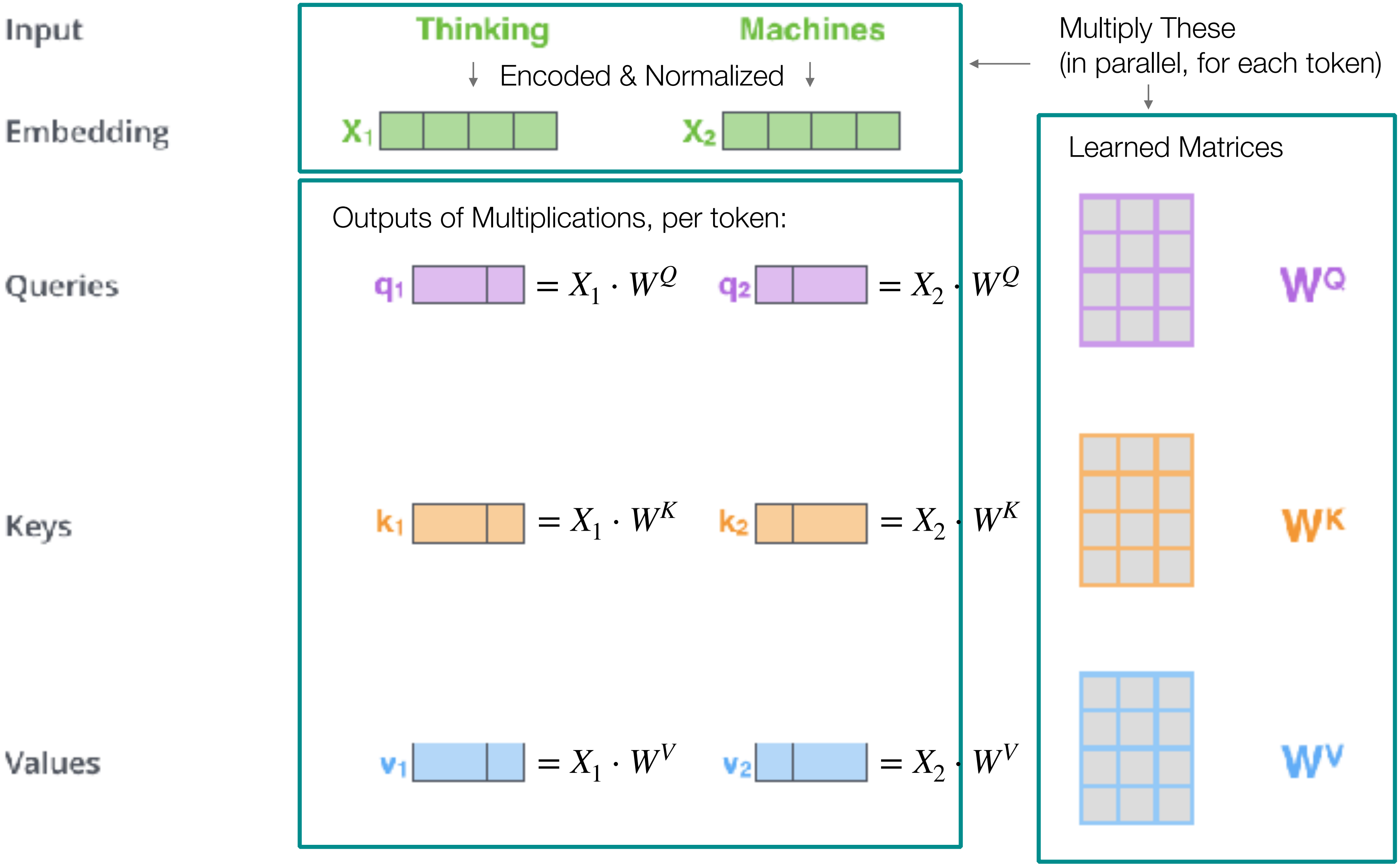


verview



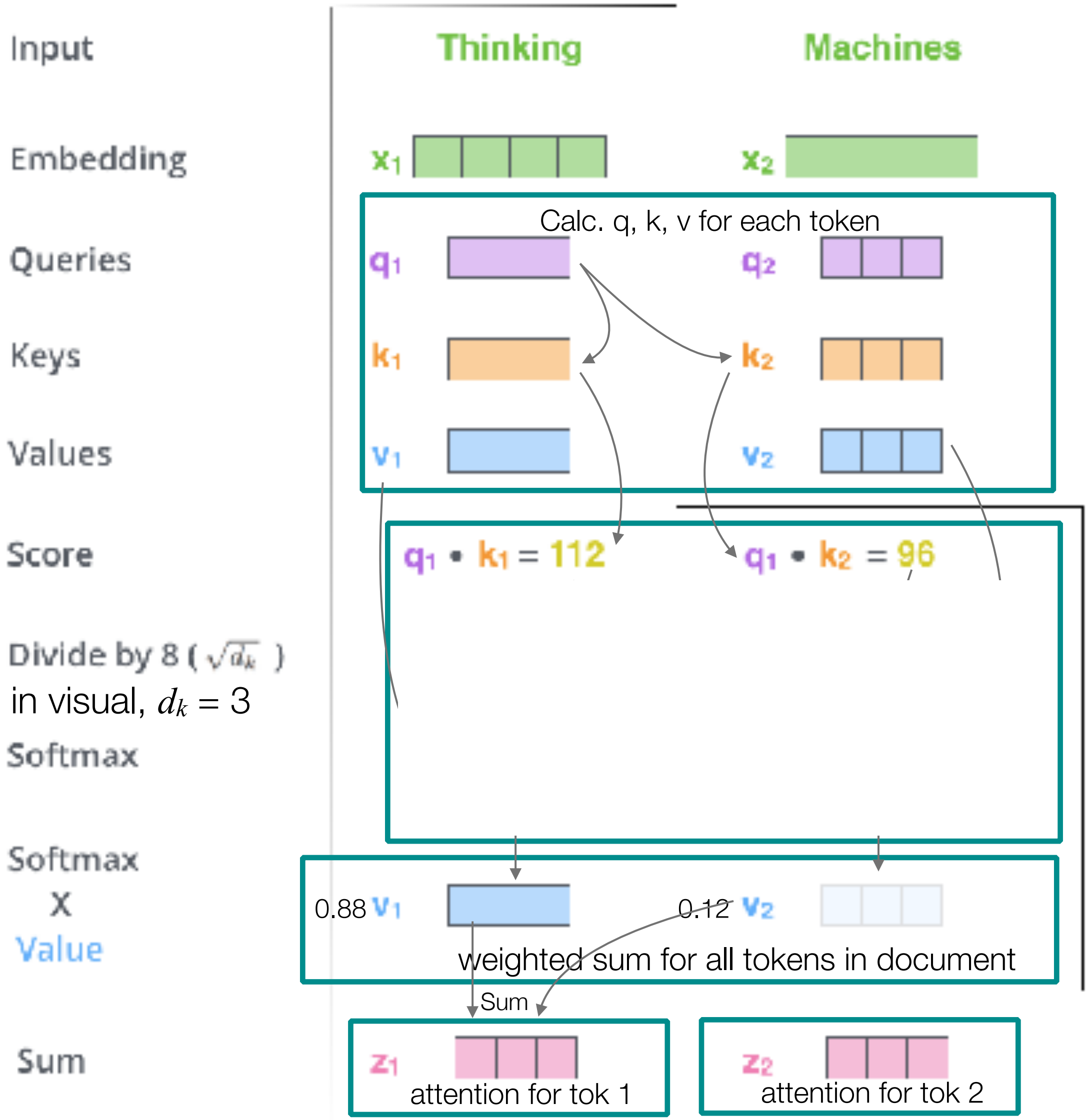
- What parameters are trained in diagram?
  - $\mathbf{W}^v, \mathbf{W}^q, \mathbf{W}^k$
- Other Parameters:
  - L: length of sequence
  - Query/Key dimension, d_k
  - Value dimension, d_v
  - Type of positional encoding (more later)
  - Cross attention versus self attention

Self Attention: in more detail

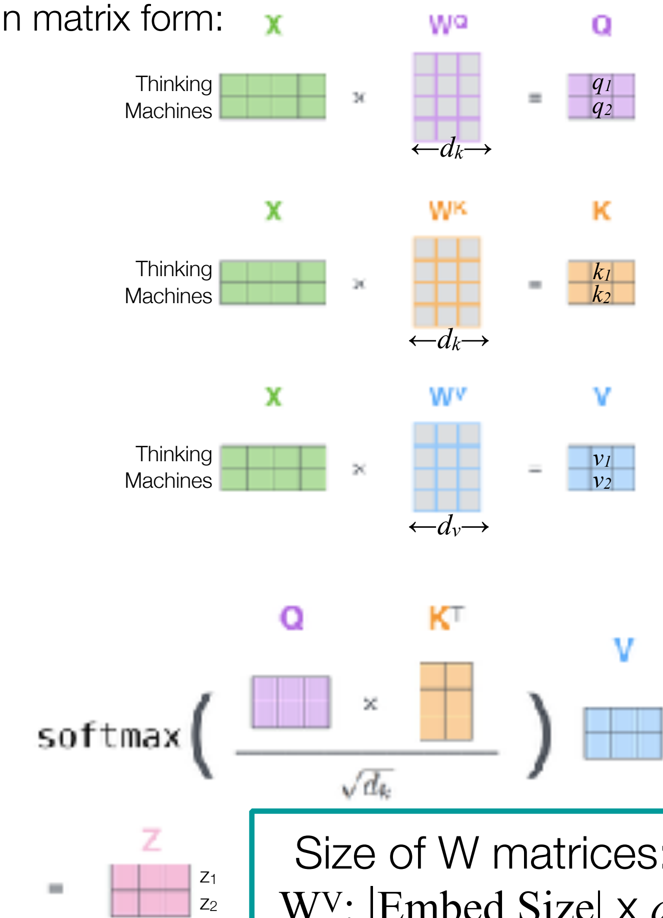


Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

Self Attention: in more detail



Straight forward to do this operation in matrix form:



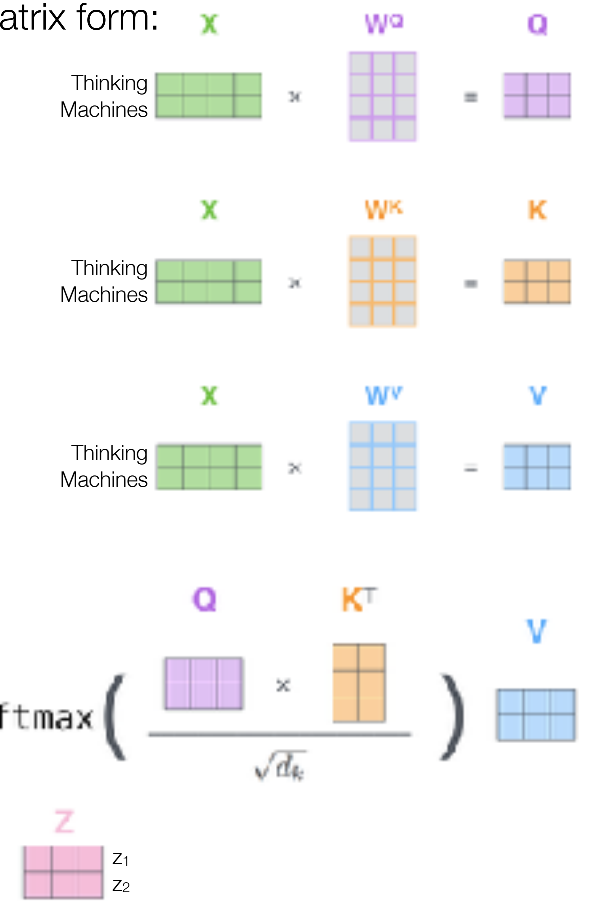
Size of W matrices:
W^V: |Embed Size| x d_v
$W^{Q,K}$: |Embed Size| x d_k

Size of Q,K,V:
|Seq Len| x d_v

Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

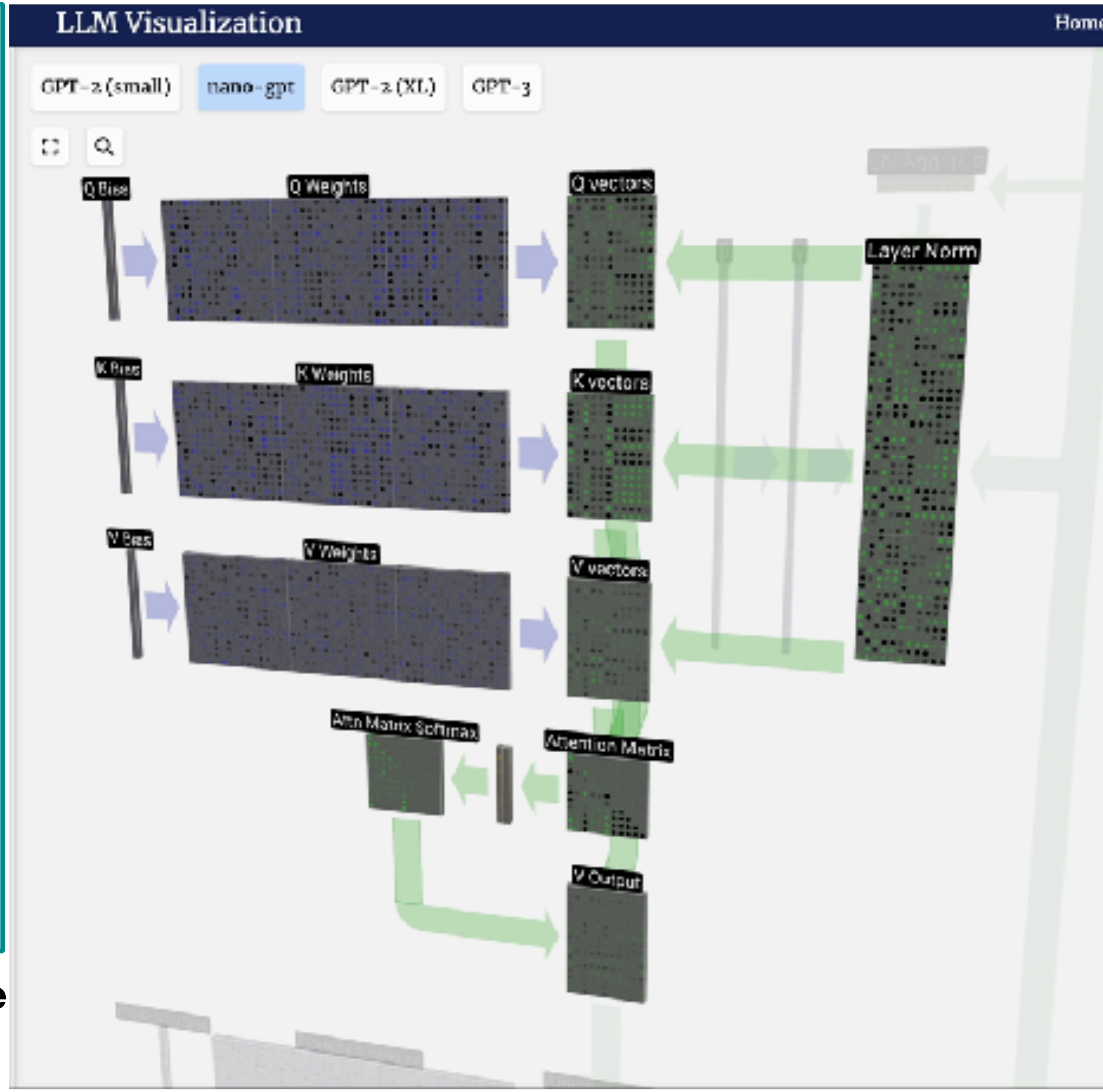
Self Attention: From https://bbycroft.net/llm

ight forward to do this operation
atrix form:



output of each head is Z_i

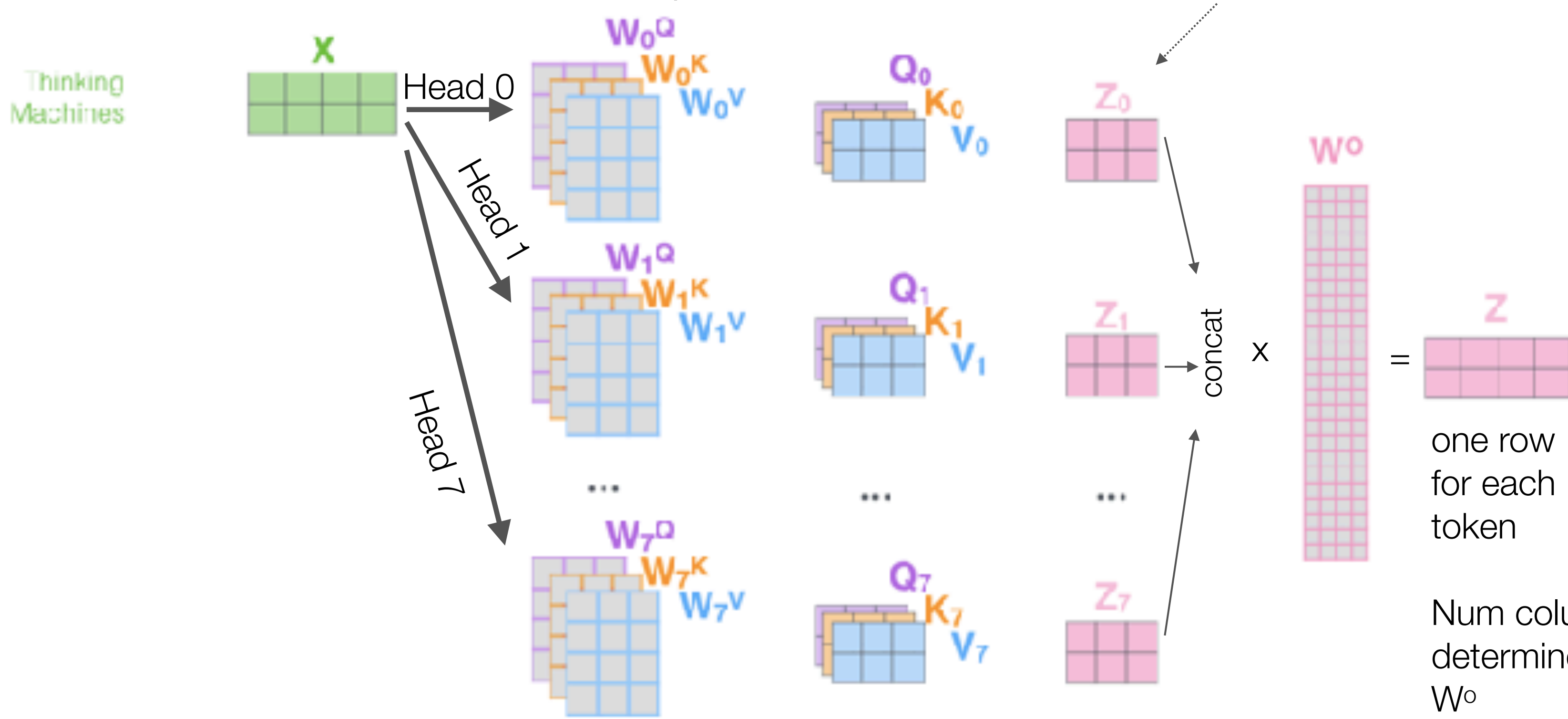
After Self-attention, **heads are processed by a FFN**



Transformer: Multi-headed Attention

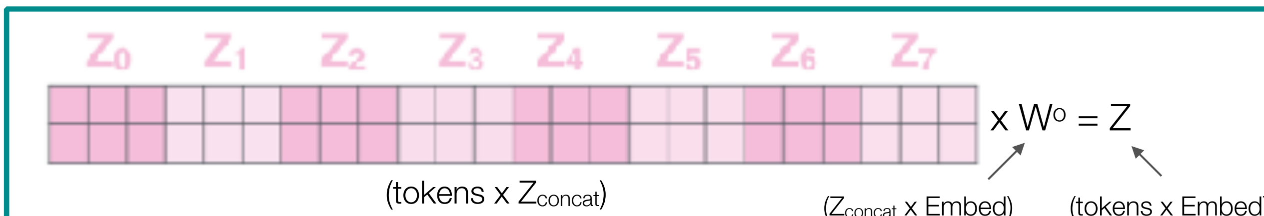
output of each head is Z_i

Z is transposed from video example



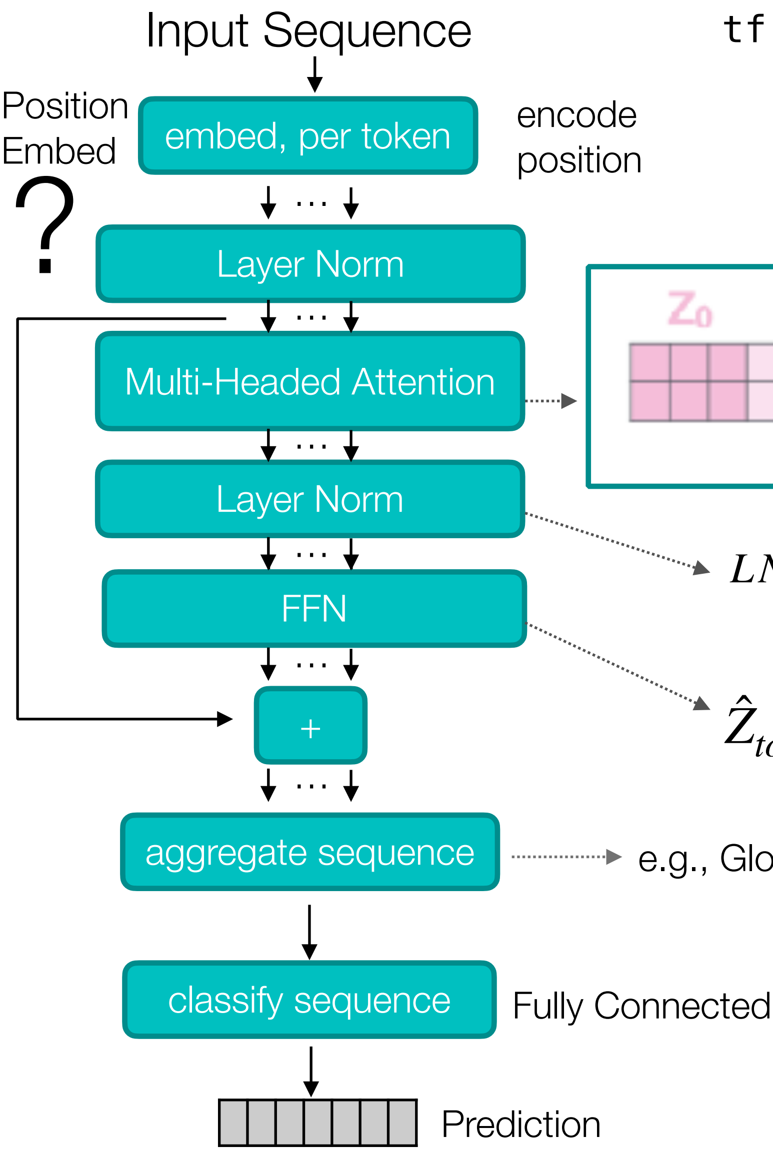
one row for each token

Num columns determined by W^O

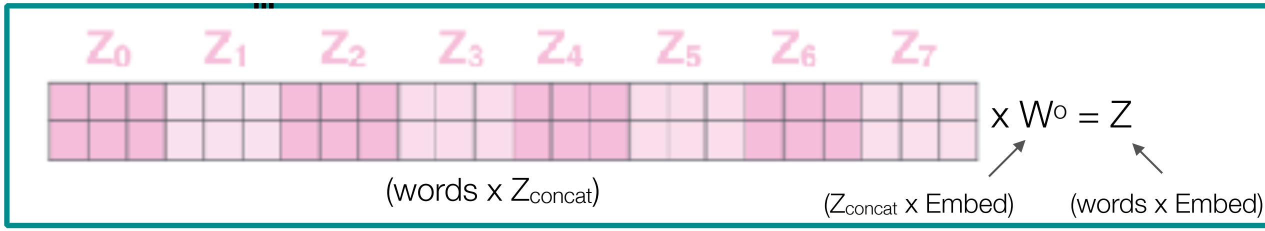


Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

Putting It Together



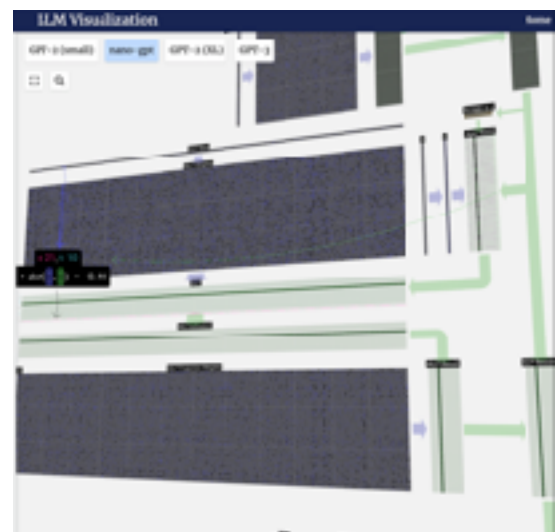

```
tf.keras.layers.MultiHeadAttention(
    num_heads,       (Number of heads Z1-Z7)
    key_dim,         (size of query/key dk)
    value_dim,       (size of each dv)
    output_shape,    (Embed size of Z, dims of Wo)
    …
```

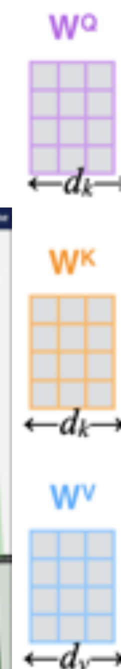


$$LN(Z) = \gamma_e \left(\frac{Z - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + \beta_e$$

Learn to normalize Z

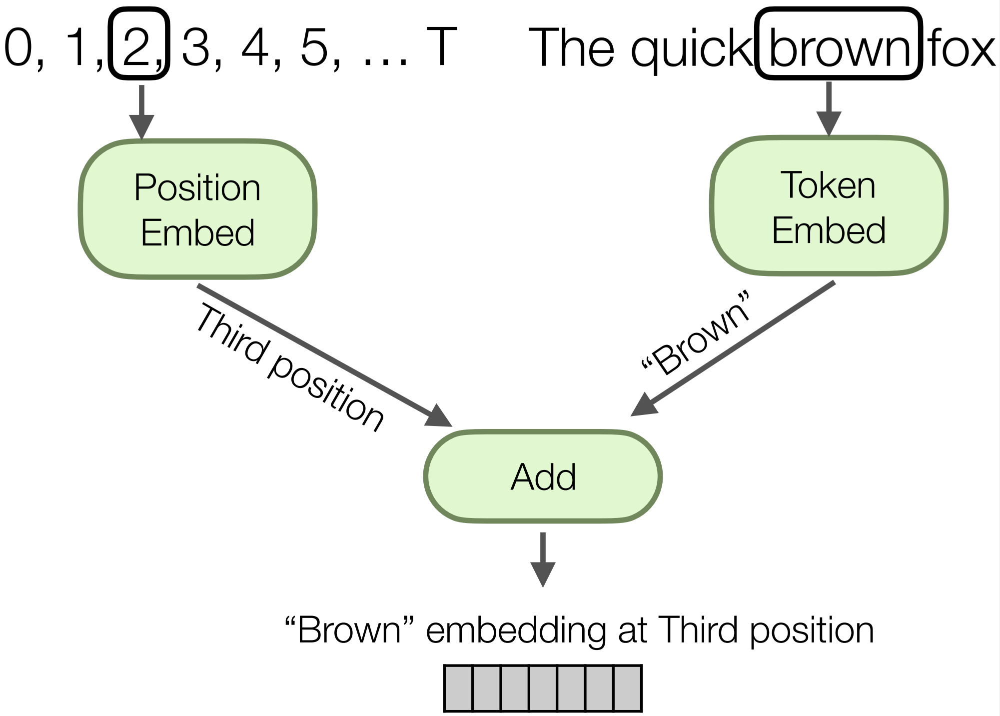
$$\hat{Z}_{tok} = W \cdot LN(Z_{tok}) + b$$

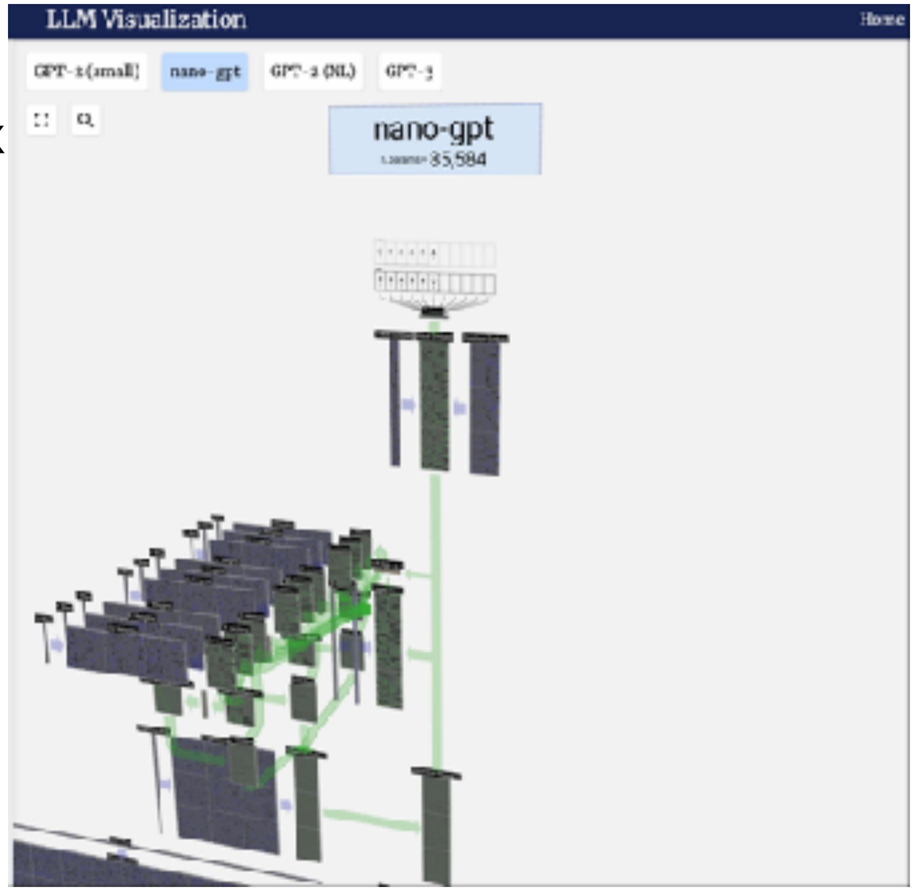




Transformer: Positional Encoding

- Objective: add notion of position to embedding
- Attempt in original paper: add sin/cos to embedding
- But could be anything that encodes position, like:





Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

Demo

The Transformer and 20 news groups with GloVe

13a. Sequence Basics [Experimental].ipynb

Lecture Notes for

Machine Learning in Python

[👨‍🏫, 👨‍💻, 🐍, 👨‍🔬]

Professor Eric Larson

Sequential CNN and Transformers