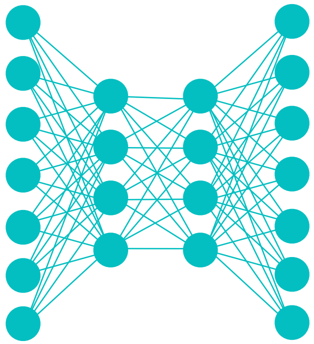
Lecture Notes for

Neural Networks and Machine Learning

Transformers

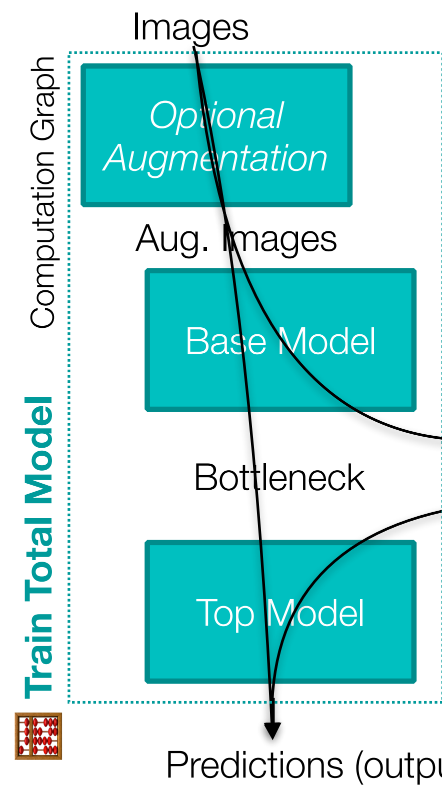


Logistics and Agenda

- Logistics
  - Paper presentations
- Agenda
  - Finish Transfer Demo
  - Transformers
- Next Time:
  - Position Encoding Transformers
  - Vision Transformers
  - Paper Presentation
  - Consistency losses

Freezing and Fine-tuning Efficiently

- **Step 1:** Freeze entire base model:
  - No update during back-propagation
  - *Optional: Augment a set of training data*
  - Send training dataset through base model
    - Save out bottleneck features
- **Step 2:** Train bottleneck features in new task
  - Typically 5-10 epochs is sufficient, easy to overfit (very fast)
  - Larger training step size is okay
- **Step 3:** Fine-tune, **unfreeze** a few layers in base model:
  - Attach newly trained model to pre-trained model, *Optional: use augmentation*
  - Train to your hearts content, use smaller training step size



Step 4: Rinse and Repeat unfreezing more layers from base model

Save

Load

Train Top Model Only

```
images = Input(shape=(IMG_SIZE, IMG_SIZE, 3))
base_model = VGG(include_top=False,
                 input_tensor=images,
                 weights="imagenet")

bottleneck = Input(shape= base_model.output.shape)
predict = Dense(NUM_CLASSES)(bottleneck)
top_model = Model(bottleneck, predict)

model_total = Model(images, outputs=
top_model(base_model.output))
```

Bottlenecking on a GPU

Dogs versus Cats

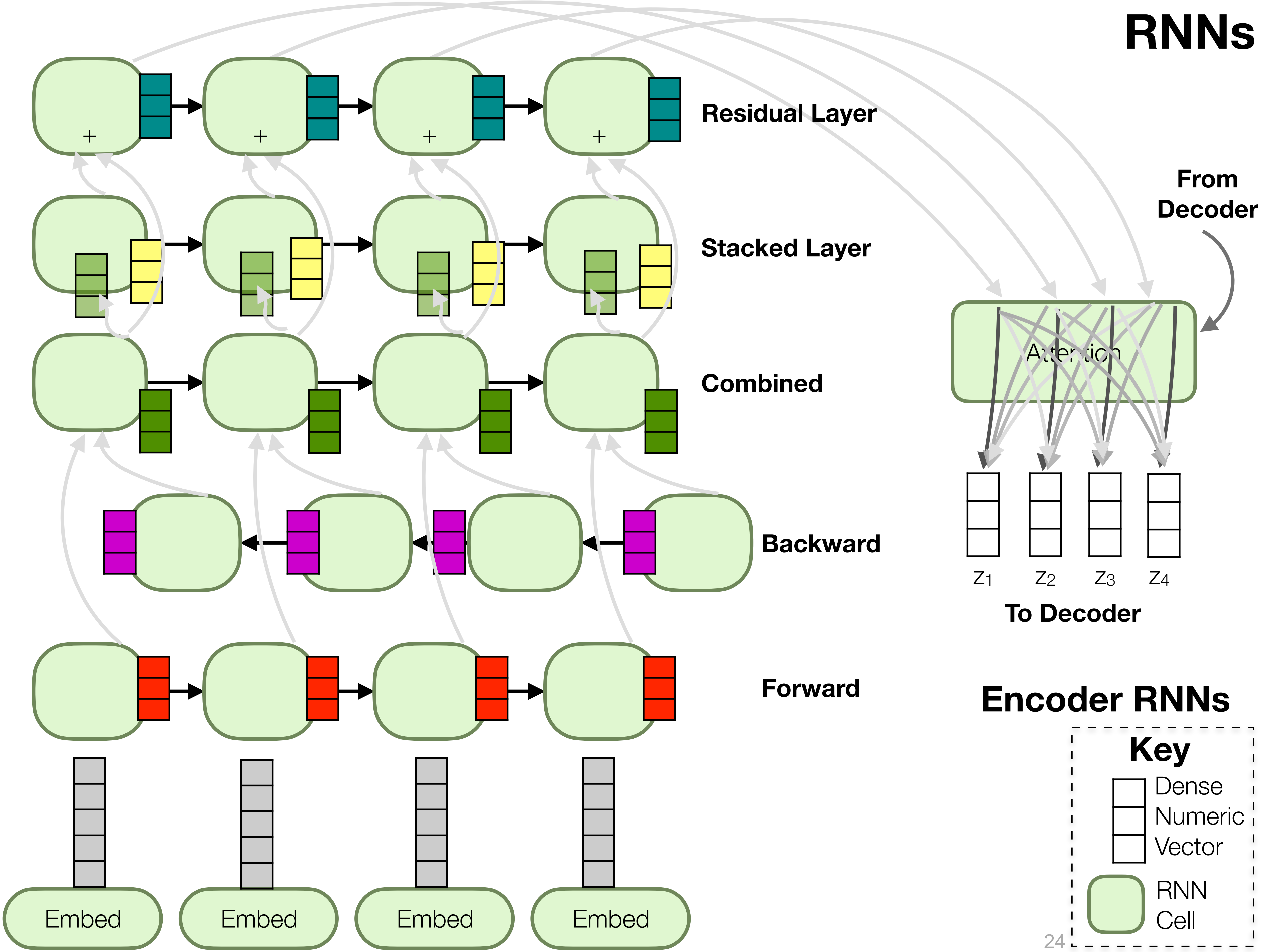
Updated for tf==2.12 in the Main Repository:
02 Transfer Learning.ipynb

Original Example: https://github.com/8000net/Transfer-Learning-Dolphins-and-Sharks

Another Great Example:
https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/

Transformers

RNNs

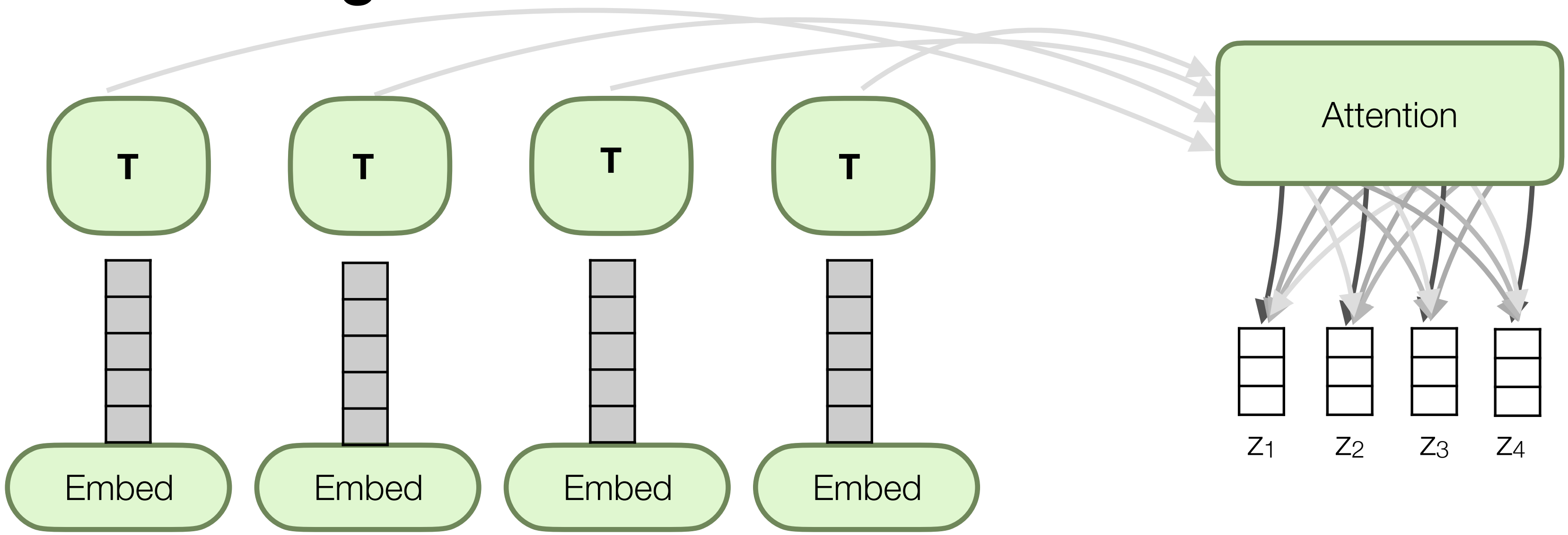


Encoder RNNs

Key
Dense Numeric Vector
RNN Cell

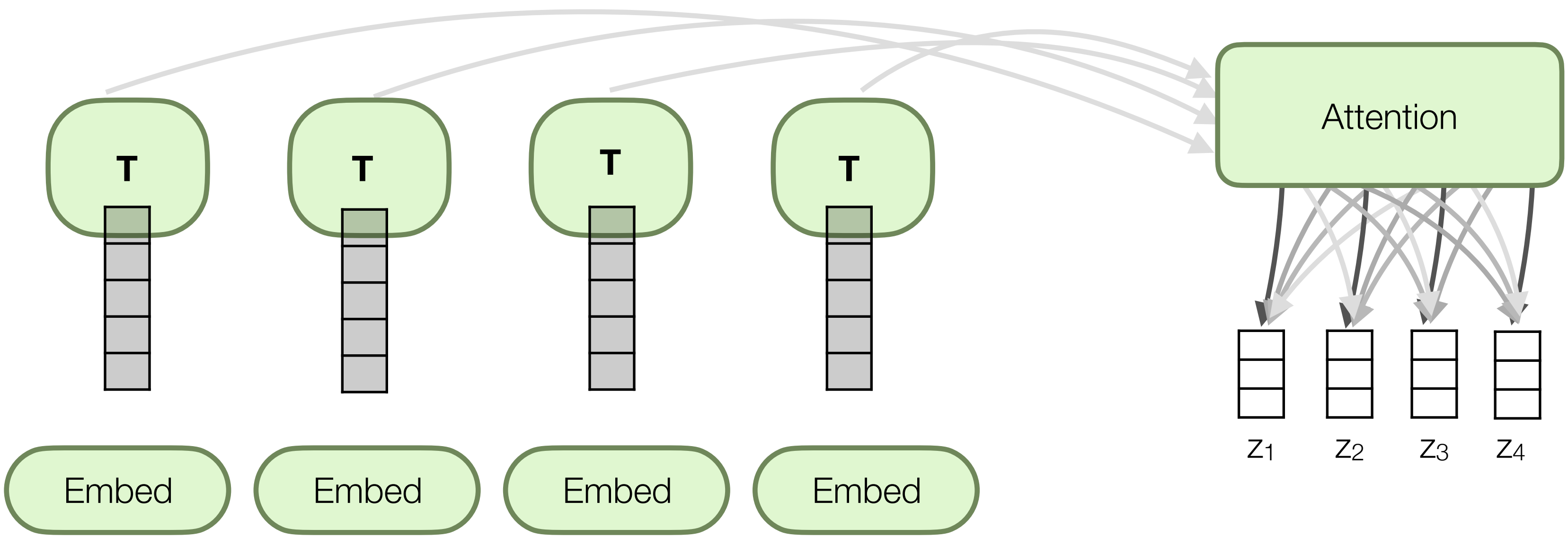
Transformers Intuition

- Recurrent networks track use an "updatable" state vector, but this takes lots of iterative processing across sequence
- 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?**



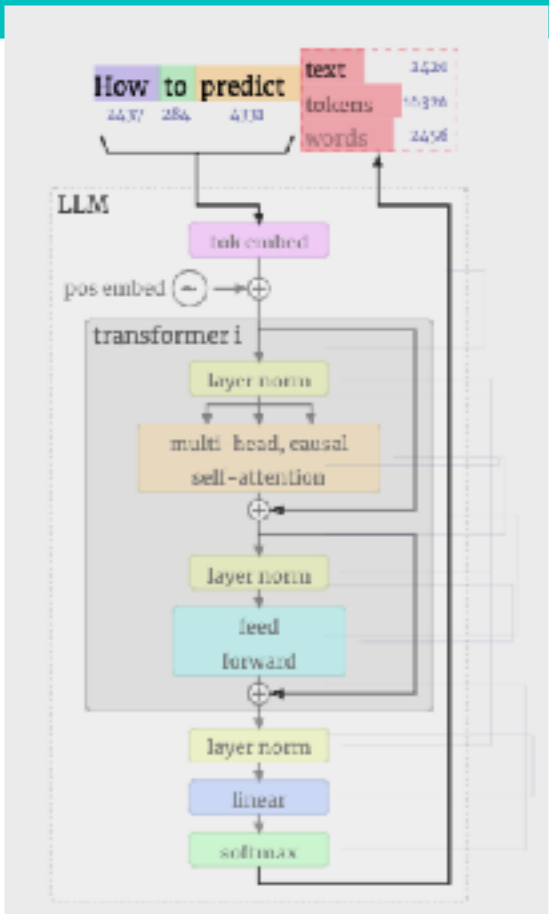
Attention is All You Need

- **Transformer Solution:**
  - Build attention into model from the **beginning**
  - Compare all words to each other through **self-attention**
  - Define a notion of "**position**"in the sequence
  - *Should capture long term relationships and be highly parallelized for GPU computing!! (**ignore memory…**)*



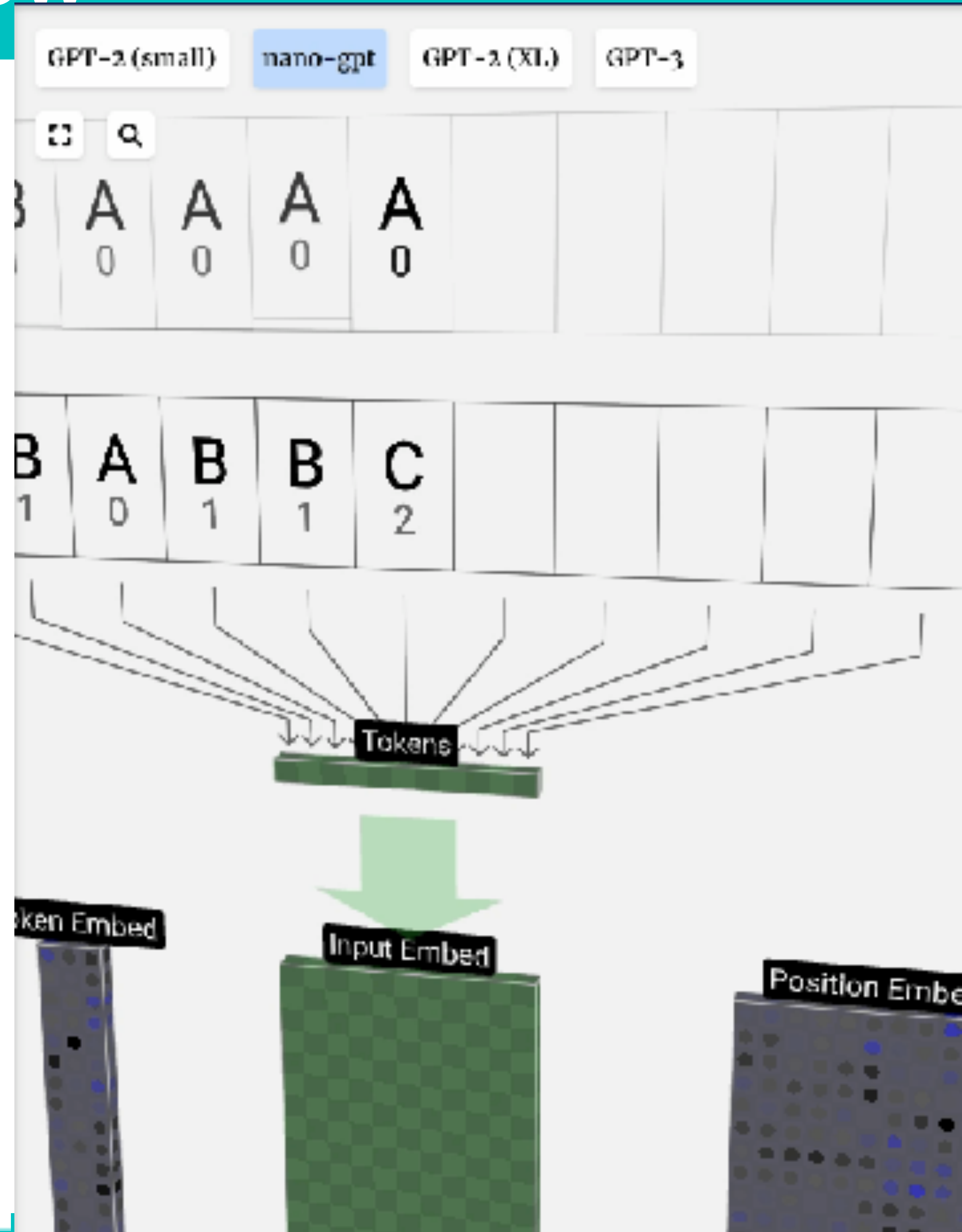
Transformer Overview

https://bbycroft.net/llm



Components

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



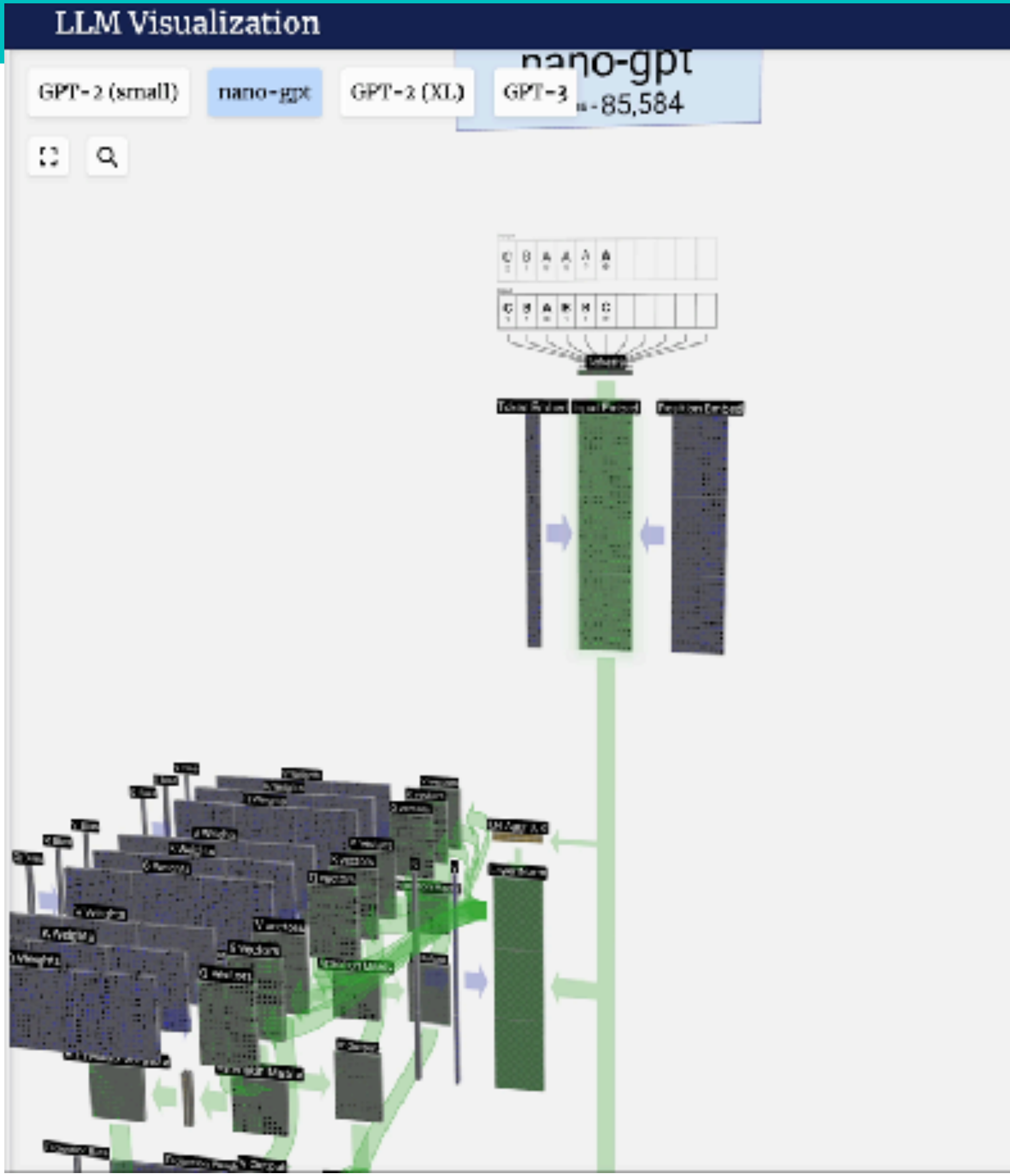
$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

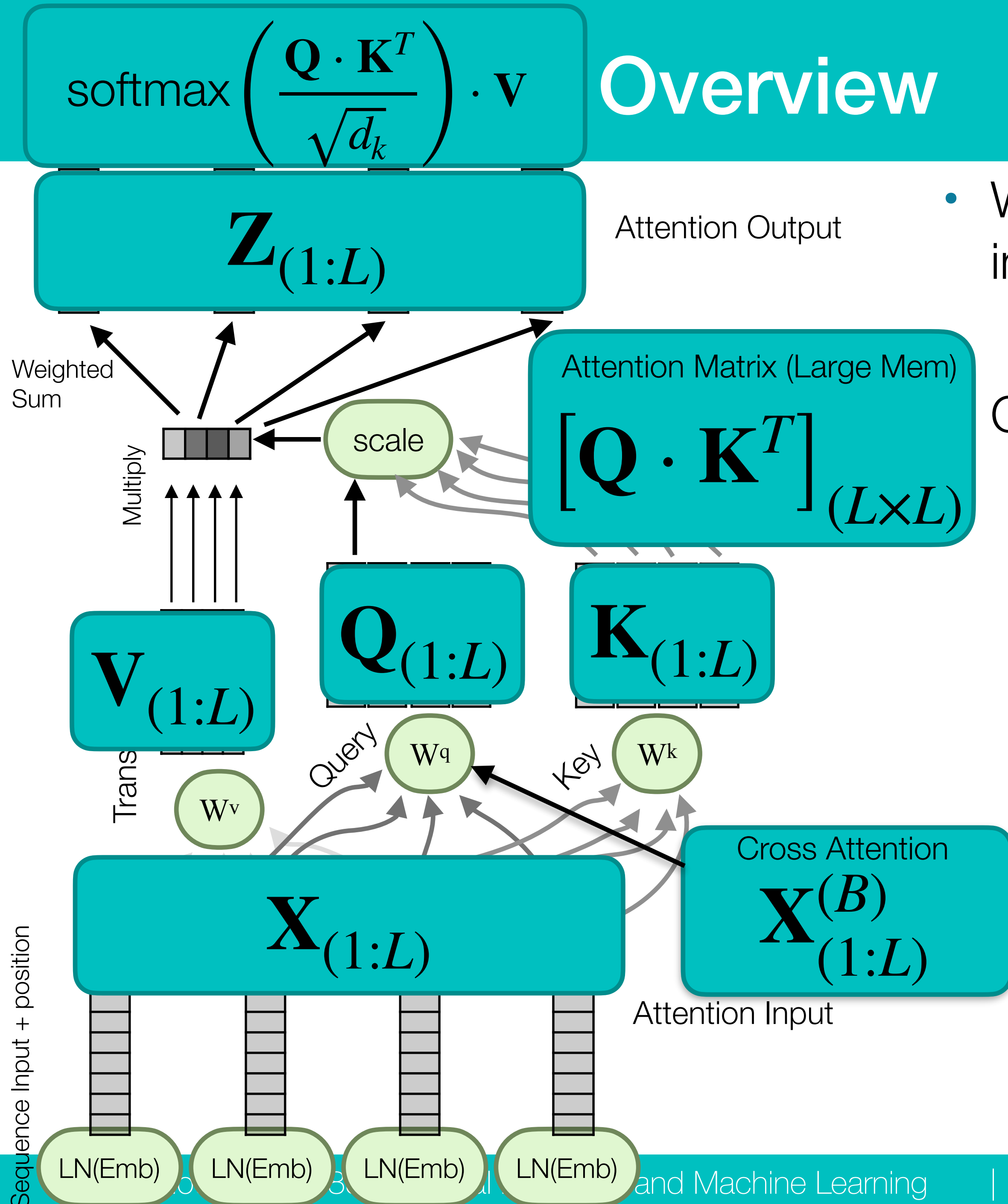
Layer Norm

Learn γ, β for each embedding column

$$LN(E^{(col)}) = \gamma_i \frac{E_i^{(col)} - \mu_E}{\sqrt{\sigma_E^2 + \epsilon}} + \beta_i$$



Overview

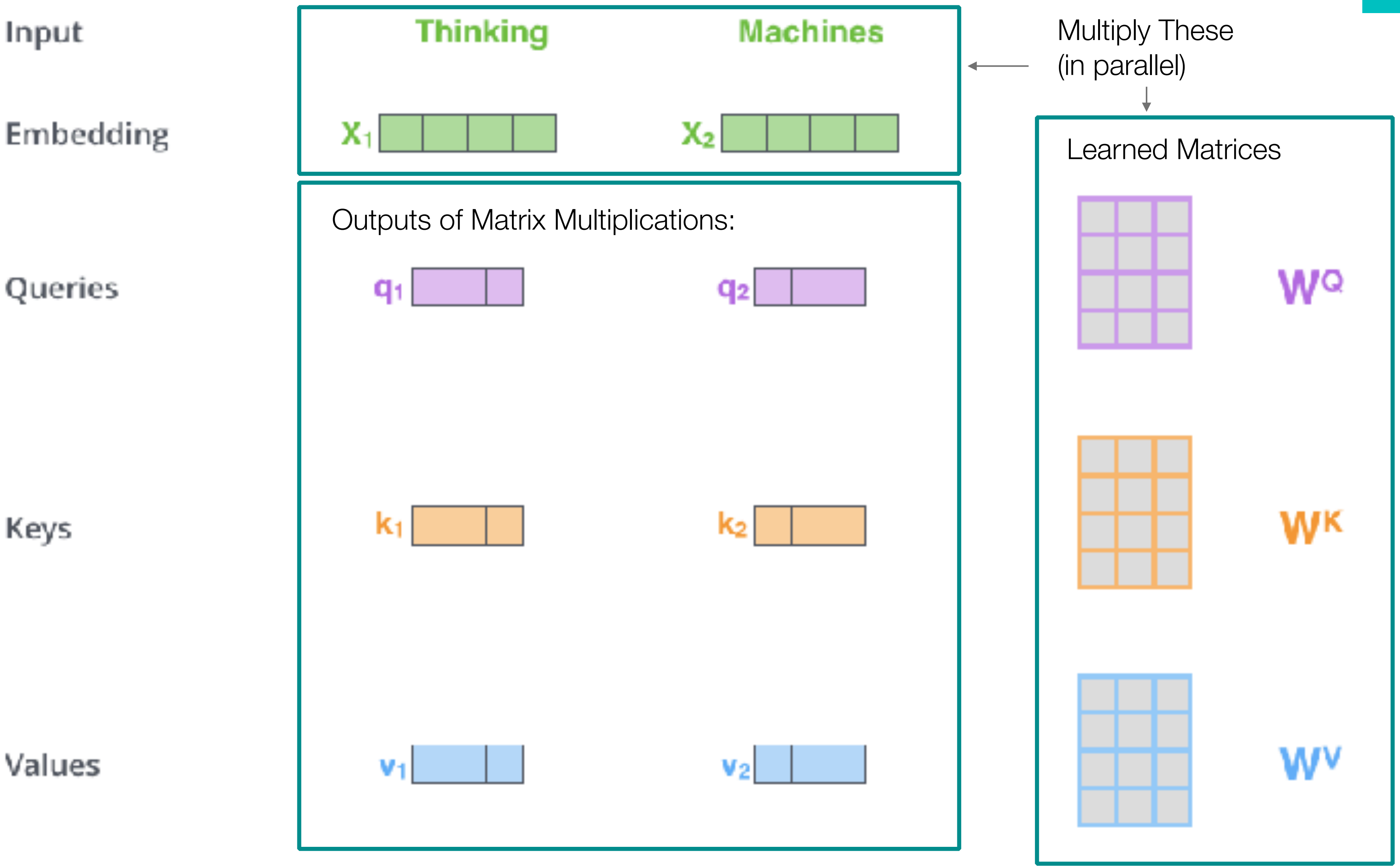


- What parameters are trained in diagram?
  - $\mathbf{W}^v, \mathbf{W}^q, \mathbf{W}^k$

Other Parameters:

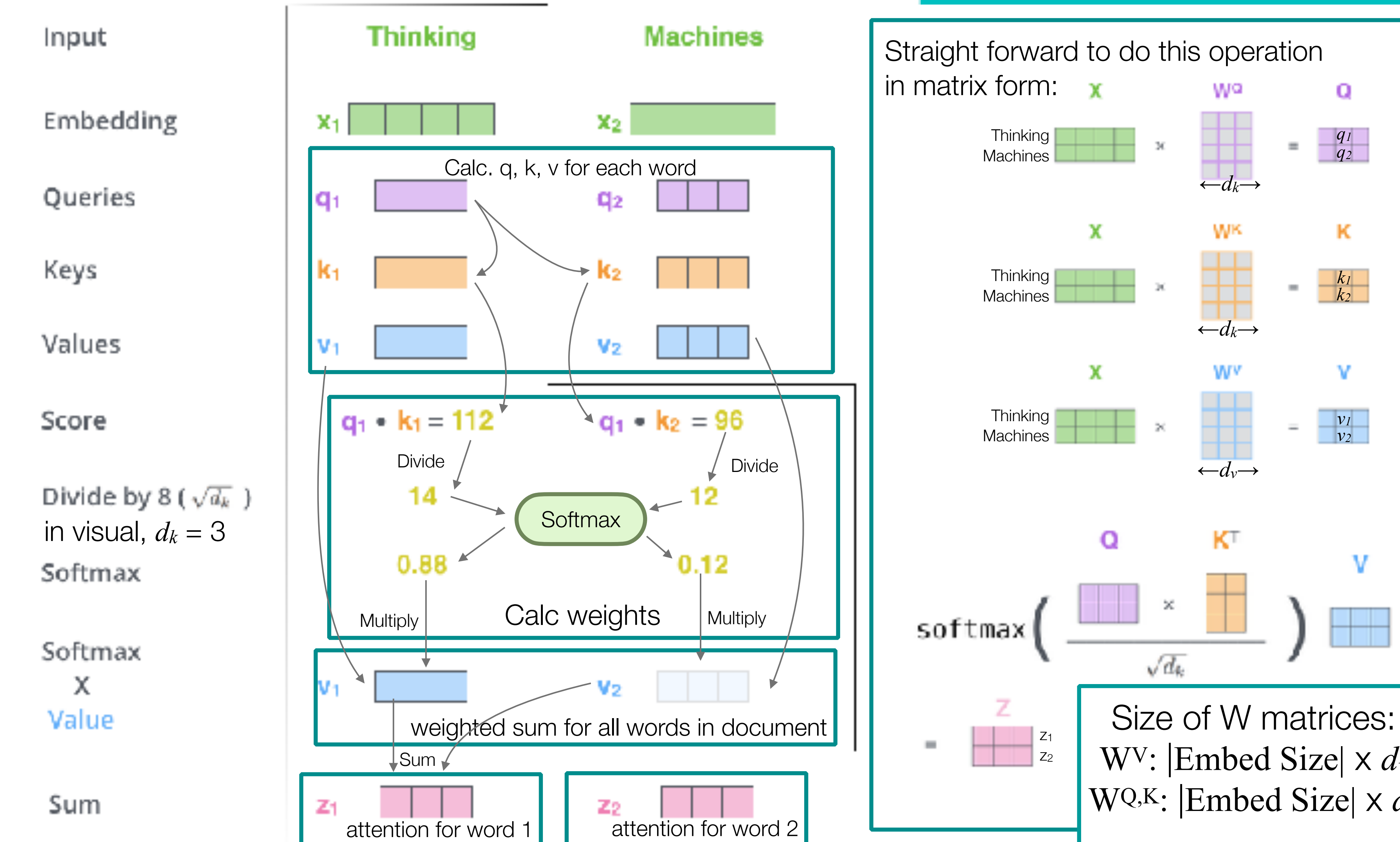
- L: length of sequence
- Query/Key dimension, d_k
- Value dimension, d_v
- Type of positional encoding (more later)
- Cross attention versus self attention

Transformer: in more detail



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

Transformer: in more detail



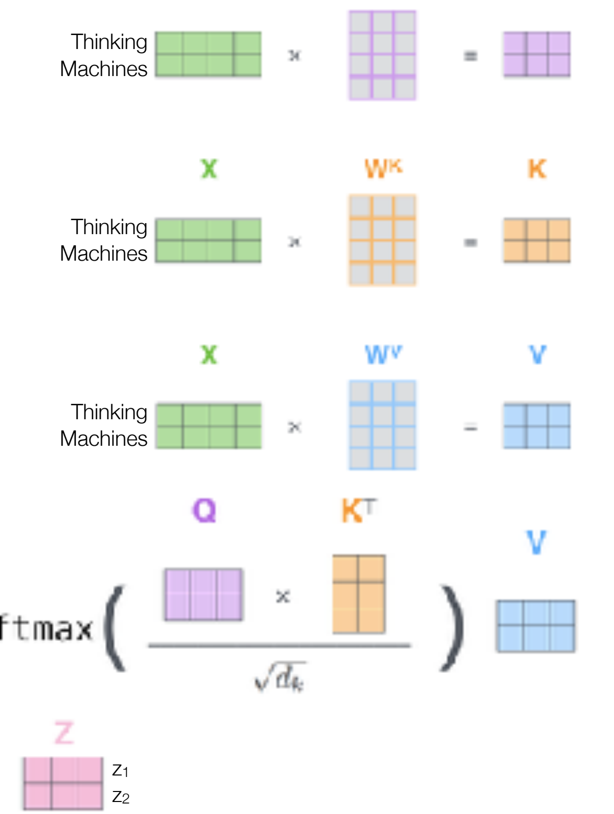
Size of W matrices:
W^V: |Embed Size| x d_v
$W^{Q,K}$: |Embed Size| x d_k

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

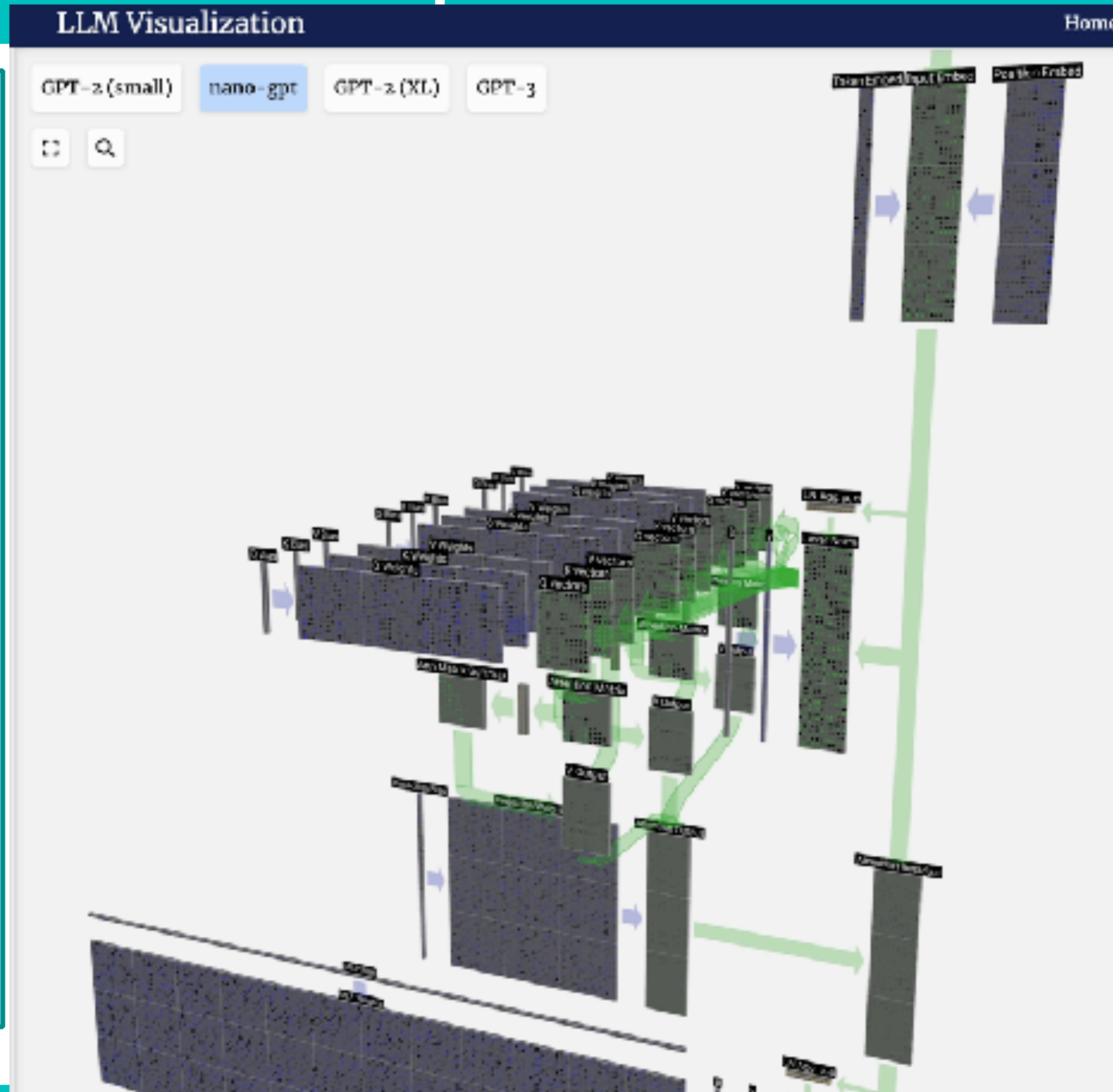
Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

Self Attention: From https://

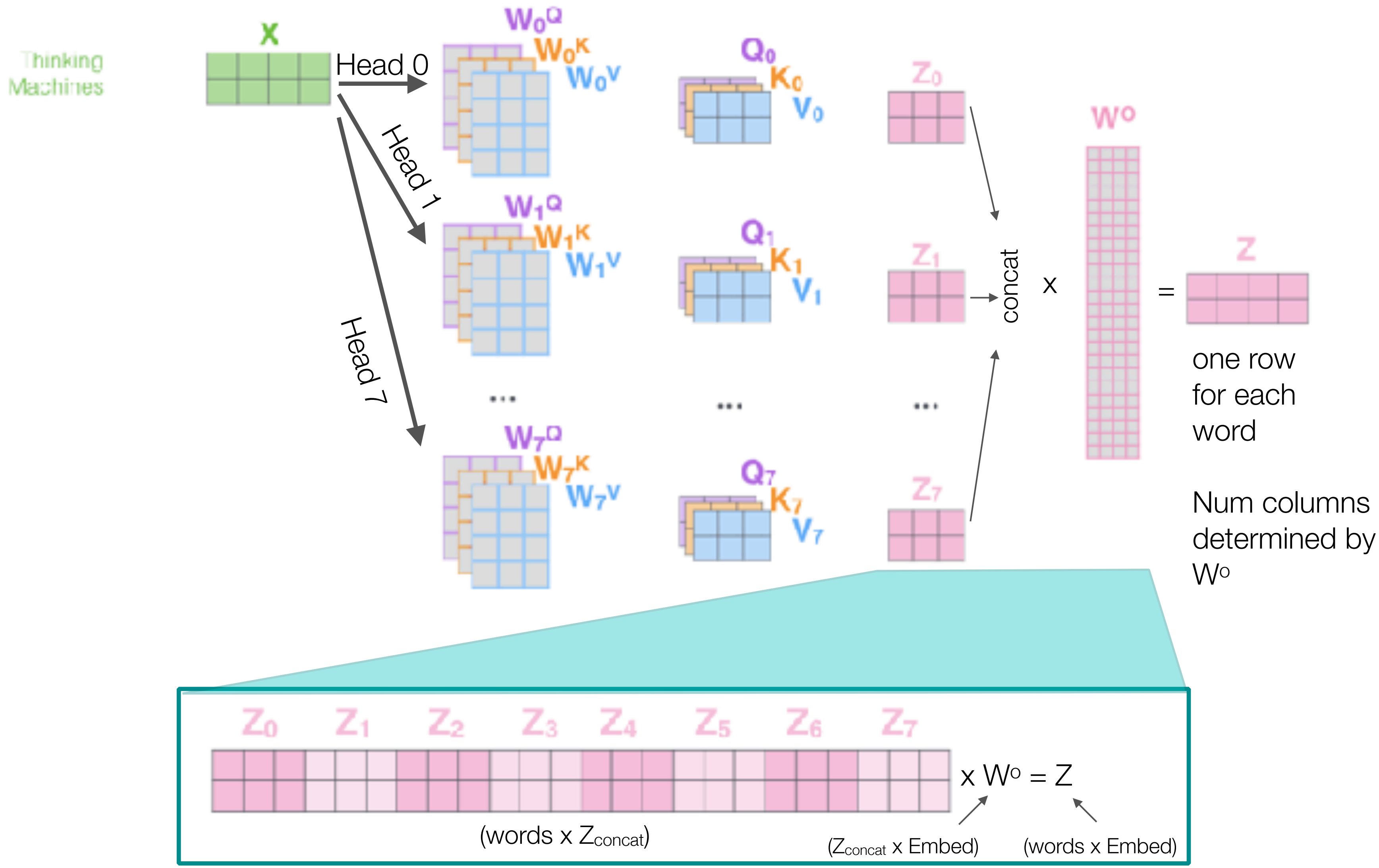
ight forward to do this operation
atrix form:



Can perform more than once!
Multiple heads!
output of each head is Z_i

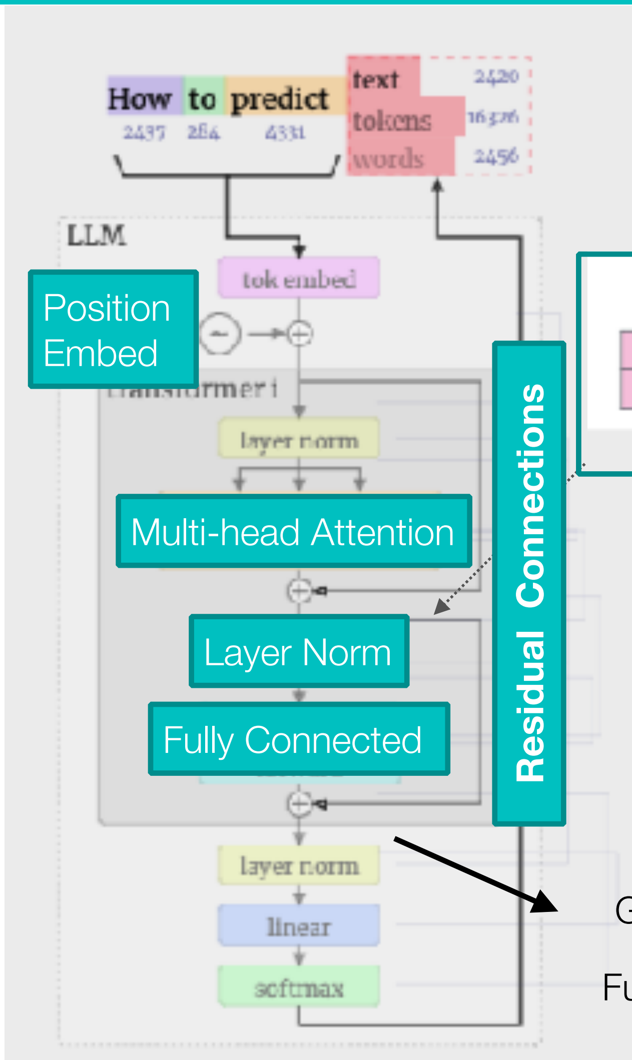


Transformer: Multi-headed Attention



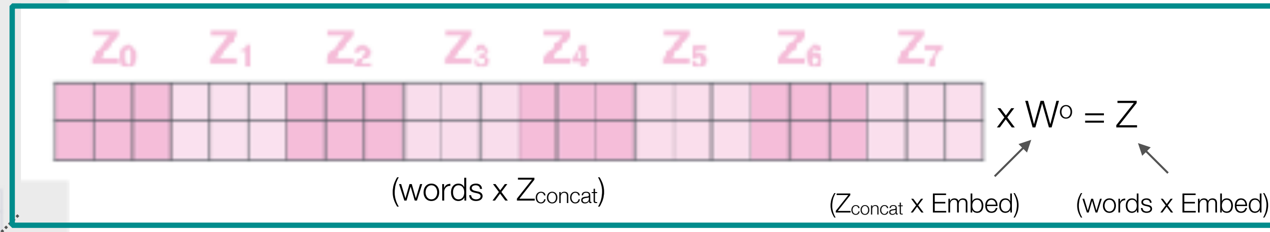
Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

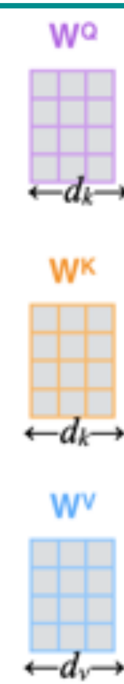
Putting It Together




```
tf.keras.layers.MultiHeadAttention(
        num_heads,
        key_dim,
        value_dim,
        output_shape,
```

(Number of heads Z_1-Z_7)
(size of query/key d_k)
(size of each d_v)
(Embed size of Z, dims of W^o)

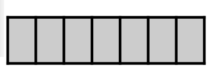




$$LN(Z_{row}) = \gamma \frac{Z_i - \mu_Z}{\sqrt{\sigma_Z^2 + \epsilon}} + \beta$$

Global Average

Fully Connected



Prediction