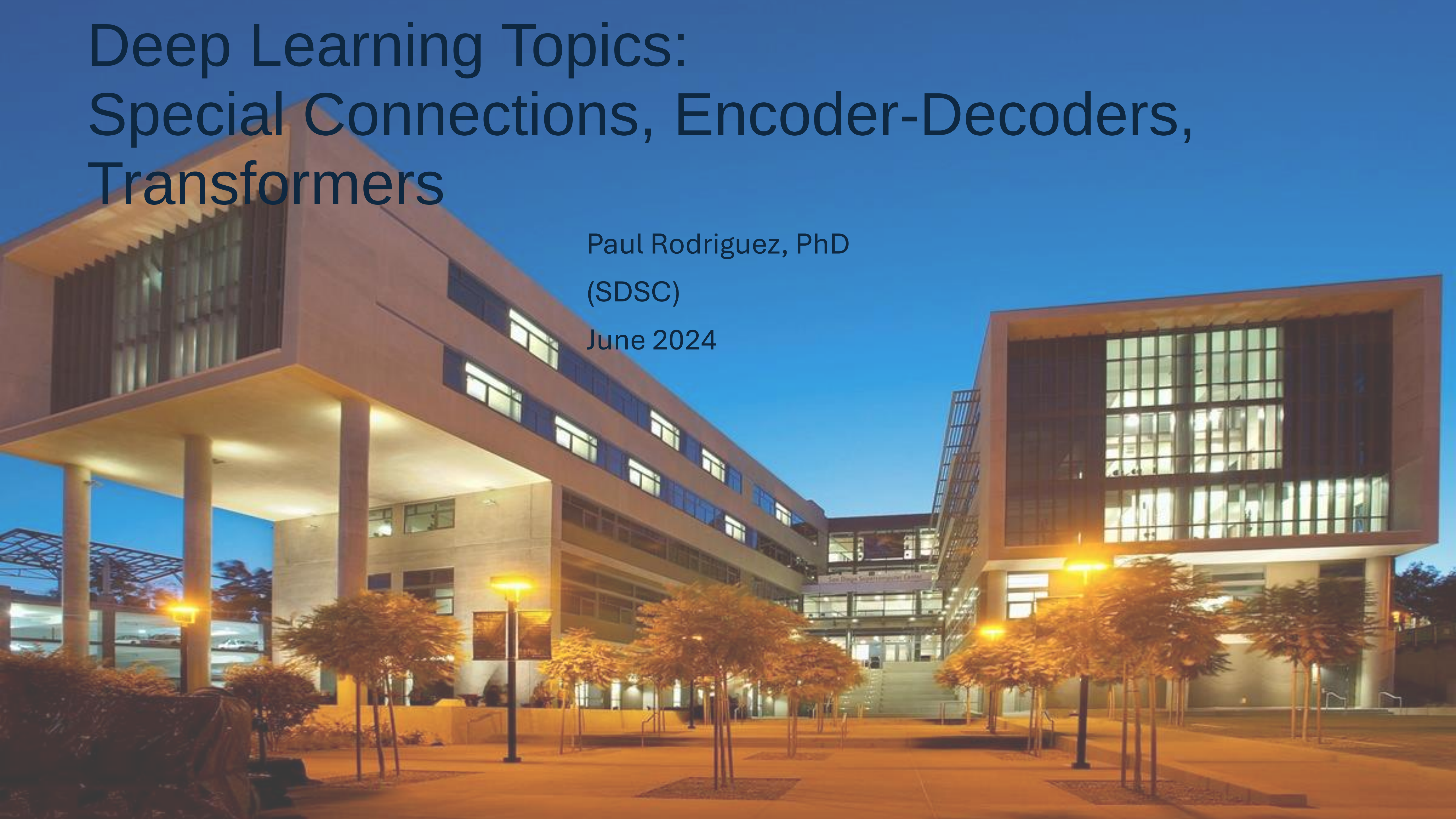
Deep Learning Topics: Special Connections, Encoder-Decoders, Transformers

Paul Rodriguez, PhD

(SDSC)

June 2024



Outline

- Basic word prediction task and motivating the attention strategy
- A basic Attention Head network and exercise
- Transformers
- BERT and GPT strategies
- Transformers in Science Applications
- Combining Images and Text

Dependences of Language

Consider this sequence:

The Law will never be perfect, but it’s application should be just - this is what we are missing, in my opinion <End of Sequence>

What does ‘it’ refer to that can have an ‘application’?

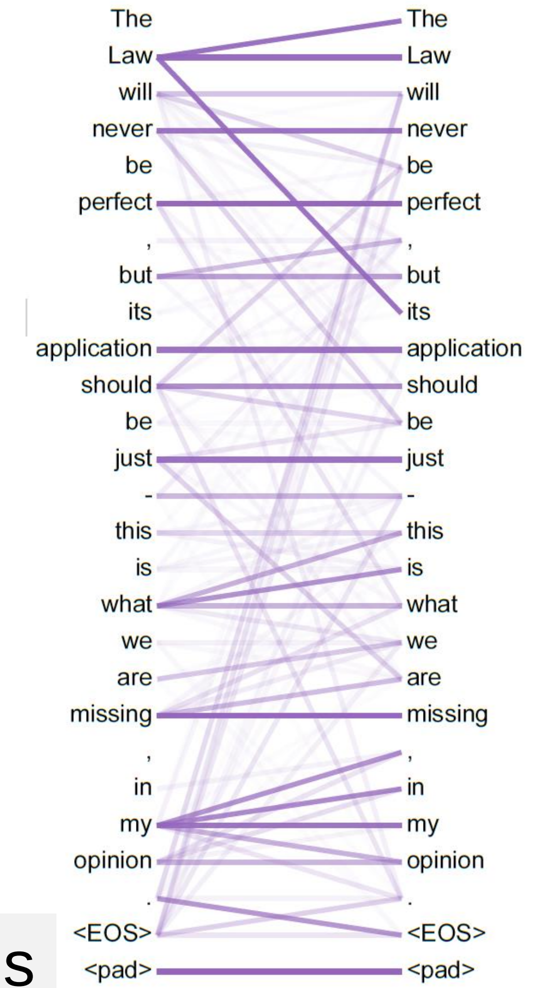
Dependences of Language

Consider this sequence:

The Law will never be perfect, but it’s application should be just - this is what we are missing, in my opinion <End of Sequence>

What does ‘it’ refer to that can have an ‘application’?

e.g ‘it’ refers back to ‘Law’, which is part of ‘the Law’ noun phrase, which is the entity that will ‘never be perfect’, and so on …



many dependencies and interactions

A toy problem to get some intuition

- Let's use the following list of 5 tokens:

  <**s**tart>, the, man, chicken, ordered

- Let's use this sequence of 6 tokens as our only:

  <start> the man ordered the chicken

- If we use **token** ids 1 to 5 it is the sequence of 6 numbers [1,2,3,5,2,4]

- **Now let's try to predict the next word by 'attention' idea**

The toy task: predict next word

The data: 5 tokens (V=5),

1 sequence (length=T=6): <Start> the man ordered the chicken

A basic solution is bigram matrix

eg a sequence of tokens (rows) and current word predictions (cols)

X= Sequence-to-Word is TxV

Pos	Word	<strt>	The	Man	Chikn	Order .
0	<start>		1.0			
1	The			0.5	0.5	
2	Man					1.0
3	Orde.r		1.0			
4	The			0.5	0.5	
5	Chick.	1.0				

The toy task: predict next word

The data: 5 tokens (V=5),
1 sequence (length=T=6): <Start> the man ordered the chicken

A basic solution is bigram matrix
eg a sequence of tokens (rows) and current word predictions (cols)

X= Sequence-to-Word is TxV

Challenge, can we learn predictions (→) that depend on context of other tokens and/or position

After <Start> the → man = 1.0

After 'Ordered' the → chicken = 1.0

Pos	Word	<strt>	The	Man	Chikn	Order.
0	<start>		1.0			
1	The			0.5	0.5	
2	Man					1.0
3	Orde.r		1.0			
4	The			0.5	0.5	
5	Chick.	1.0				

The attention idea

Let's get all tokens to 'pass information' about dependencies

E.G. for *X a TxV* matrix, we want *W a TxT* matrix – aka 'attention' weights - so that *W*X* has contextual predictive information

W dependencies is TxT **X= Sequence-to-Word is TxV**

$$\begin{pmatrix} w_{11} & \cdots & w_{1T} \\ \vdots & \vdots & \vdots \\ w_{T1} & \cdots & w_{TT} \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \end{pmatrix} \rightarrow\rightarrow$$

The attention idea

Let's get all tokens to 'pass information' about dependencies

E.G. for *X a TxV* matrix, we want *W a TxT* matrix – aka 'attention' weights - so that *W*X* has contextual predictive information

W dependencies is TxT **X= Sequence-to-Word is TxV**

$$\begin{pmatrix} w_{11} & \cdots & w_{1T} \\ \vdots & \vdots & \vdots \\ w_{T1} & \cdots & w_{TT} \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ [illegible] & [illegible] & [illegible] & [illegible] & 0 \end{pmatrix} \rightarrow \rightarrow$$

W will be learned and should also reflect the sequence interdependencies

The attention idea

Finally, apply Softmax to each output row to make it like probability scores (ie our predictions)

W dependencies is TxT **X= Sequence-to-Word is TxV**

$$\begin{pmatrix} w_{11} & \cdots & w_{1T} \\ \vdots & \vdots & \vdots \\ w_{T1} & \cdots & w_{TT} \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ 1.0 & 0 & 0 & 0 & 0 \end{pmatrix} \rightarrow\rightarrow \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Making predictions causal
(ie only depends on previous tokens)

First build a TxT mask so that sequence position t only uses columns 1:t

$$Mask = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Then, multiply it by *W_dep* matrix

Now multiply masked dependency elementwise to W

$$W_{dep} \odot Mask = \begin{pmatrix} w_{11} & 0 & \cdots & \cdots & 0 \\ w_{21} & w_{22} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{T1} & w_{T2} & w_{T3} & \cdots & w_{TT} \end{pmatrix}$$

Building (intuition of) an Attention Network

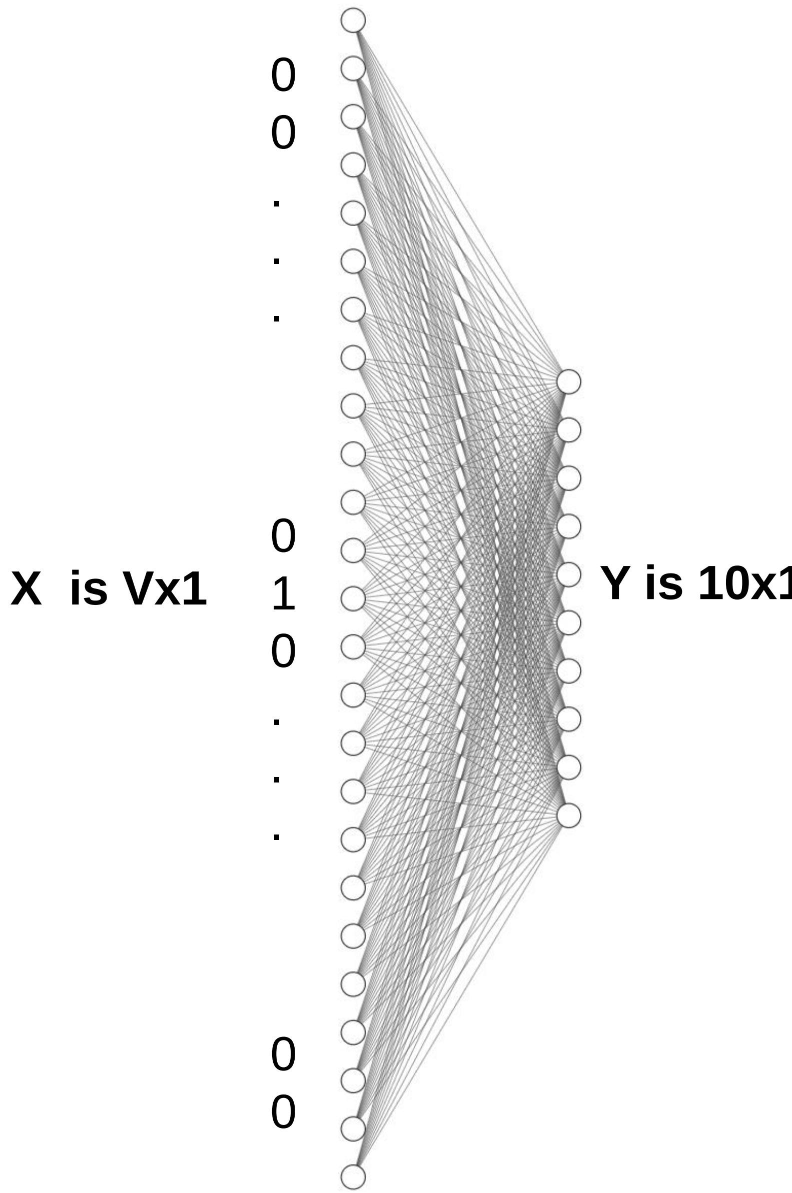
Convert text to input sequence of token ids (tokenize)

Use simple embedding layers to process input

Transform embeddings into “Query”, “Key” matrices

Let Q x K = attention matrix that represents interdependencies

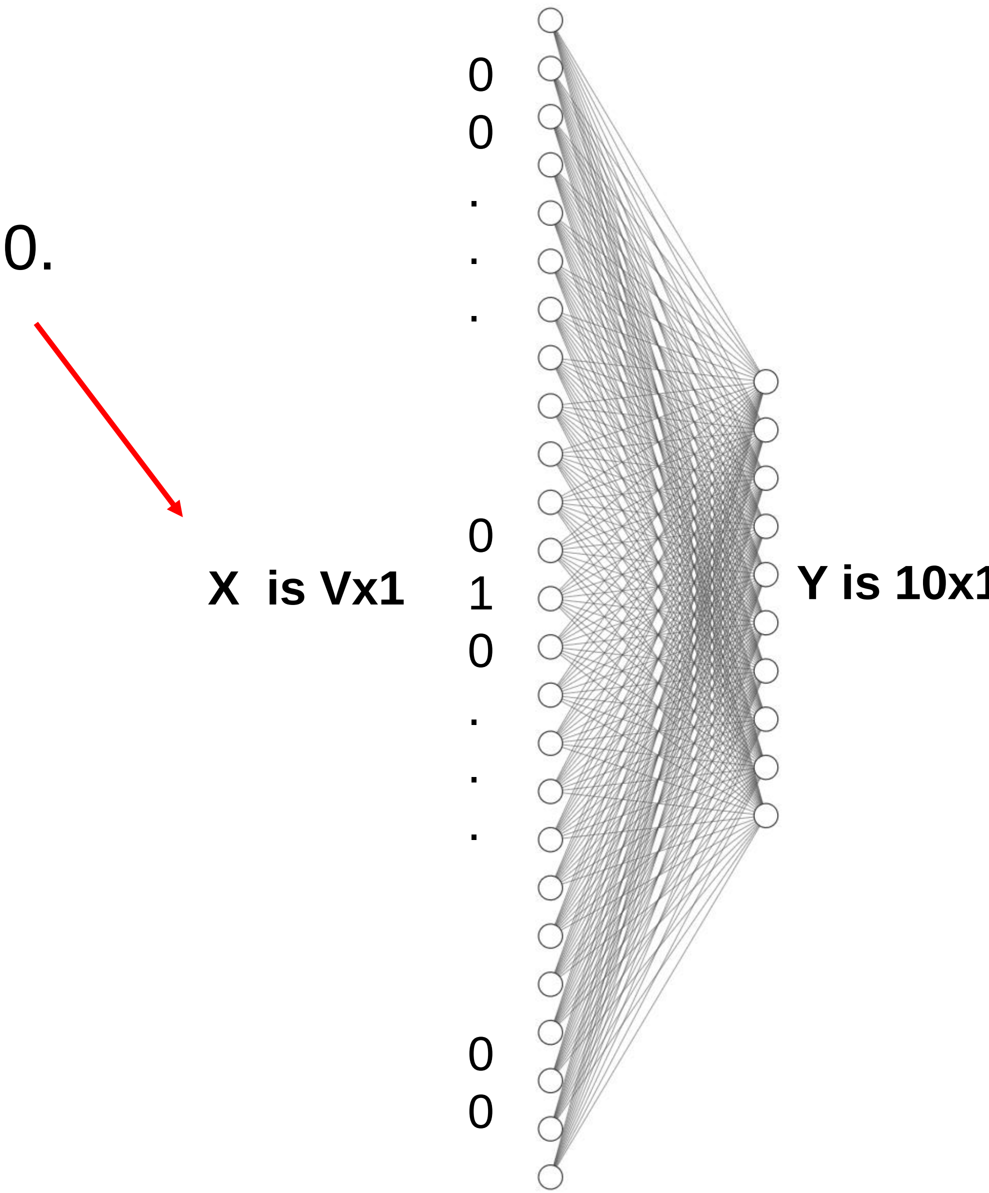
An Embedding Layer/Table



An Embedding Layer/Table

Let *X* have one of the V input nodes = 1, the rest are 0.

Use linear activation so that $Y = W * X$



An Embedding Layer/Table

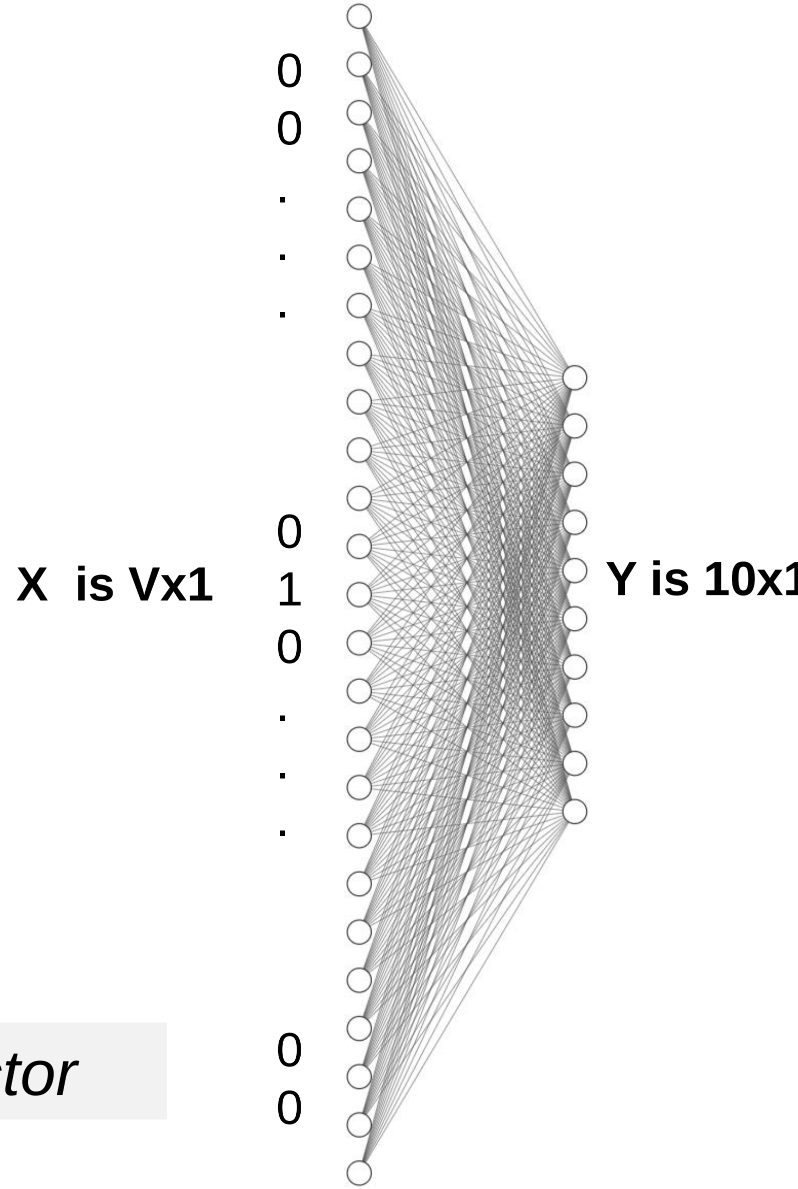
Let *X* have one of the V input nodes = 1, the rest are 0.

Use linear activation so that $Y = W * X$

If j-th node is 1, then the Y output is just the jth col of *W*

$$Y = W * X = W_j$$

Thus, each token id is converted to a lower dimensional vector



An Attention Head construction

1. **Let X = sequence of token ids**

For embedding dimension E, map: $\underset{T\times 1}{\mathrm{X}} \rightarrow \underset{T\times E}{Xemb}$

For positions 1...T map: $\underset{T\times 1}{\mathrm{P}} \rightarrow \underset{T\times E}{Xpos}$

Take as new input: $X = Xemb + Xpos$

An Attention Head construction

1. **Let X = sequence of token ids**

For embedding dimension E, map: $\underset{T\times 1}{\mathrm{X}} \rightarrow \underset{T\times E}{Xemb}$

For positions 1...T map: $\underset{T\times 1}{\mathrm{P}} \rightarrow \underset{T\times E}{Xpos}$

Take as new input: $X = Xemb + Xpos$

2. For dimension H, get Q,K,V matrices: $\underset{T\times E}{\mathrm{X}} \rightarrow \underset{T\times H}{Query}$ $\underset{T\times E}{\mathrm{X}} \rightarrow \underset{T\times H}{Key}$ $\underset{T\times E}{\mathrm{X}} \rightarrow \underset{T\times H}{Value}$

An Attention Head construction

1. **Let X = sequence of token ids**

For embedding dimension E, map: $\underset{T\times 1}{\mathrm{X}} \rightarrow \underset{T\times E}{Xemb}$

For positions 1...T map: $\underset{T\times 1}{\mathrm{P}} \rightarrow \underset{T\times E}{Xpos}$

Take as new input: $X = Xemb + Xpos$

2. For dimension H, get Q,K,V matrices: $\underset{T\times E}{\mathrm{X}} \rightarrow \underset{T\times H}{Query}$ $\underset{T\times E}{\mathrm{X}} \rightarrow \underset{T\times H}{Key}$ $\underset{T\times E}{\mathrm{X}} \rightarrow \underset{T\times H}{Value}$

3. Get Attention Weights and Output: $\underset{T\times H}{\mathrm{Q}} * \underset{H\times T}{\mathrm{K}'} \rightarrow \underset{T\times T}{\mathrm{W}_{\mathrm{dep}}}$

$$Output = softmax(W_{dep} \odot Mask) * V$$

All mappings are linear transformations

An Attention Head code

Hyperparameters:

E is for embedding dimension,

H is for “head size” (dimension) of transformations (we could let E = H)

11 Layers/Functions

```
#Now build model to learn transformation for Q,K,V matrices

Xsequence      = tf.keras.layers.Input(shape=(T,V)) #the batch size is left unspecified
Pos_Input      = tf.keras.layers.Input(shape=(T)) #just the t=1...T integer
Pos_Embed      = tf.keras.layers.Embedding(T,V, input_length=T,name='PosEmbed')(Pos_Input) #input wi
Xinputs        = tf.keras.layers.Add()([Xsequence, Pos_Embed])

#now feed to Q,K,V transformations
Qmat        = tf.keras.layers.Dense(H,activation='linear',use_bias=False,name='Qmat')(Xinputs) #so f
Kmat        = tf.keras.layers.Dense(H,activation='linear',use_bias=False,name='Kmat')(Xinputs)
Vmat        = tf.keras.layers.Dense(H,activation='linear',use_bias=False,name='Vmat')(Xinputs)

#now apply QtoK take softmax, scale it , apply to V
QK           = tf.keras.layers.Dot(axes=(2))([Qmat,Kmat])  #it will treat each Batch item separately
QKscaled     = tf.keras.layers.Lambda(lambda x: x * scale_constant)(QK)    #for each x in QK mult by
Attn_Wts      = tf.keras.layers.Softmax(axis=2,name='AttnWts')(QKscaled, mask=Msk)        #apply mas
Vout        = tf.keras.layers.Dot(axes=1,name='Voutput')([Attn_Wts,Vmat])

my_attn_model    = tf.keras.Model(inputs = [Xsequence,Pos_Input], outputs=Vout)
```

An Attention Head code

Hyperparameters:

E is for embedding dimension,

H is for “head size” (dimension) of transformations (we could let E = H)

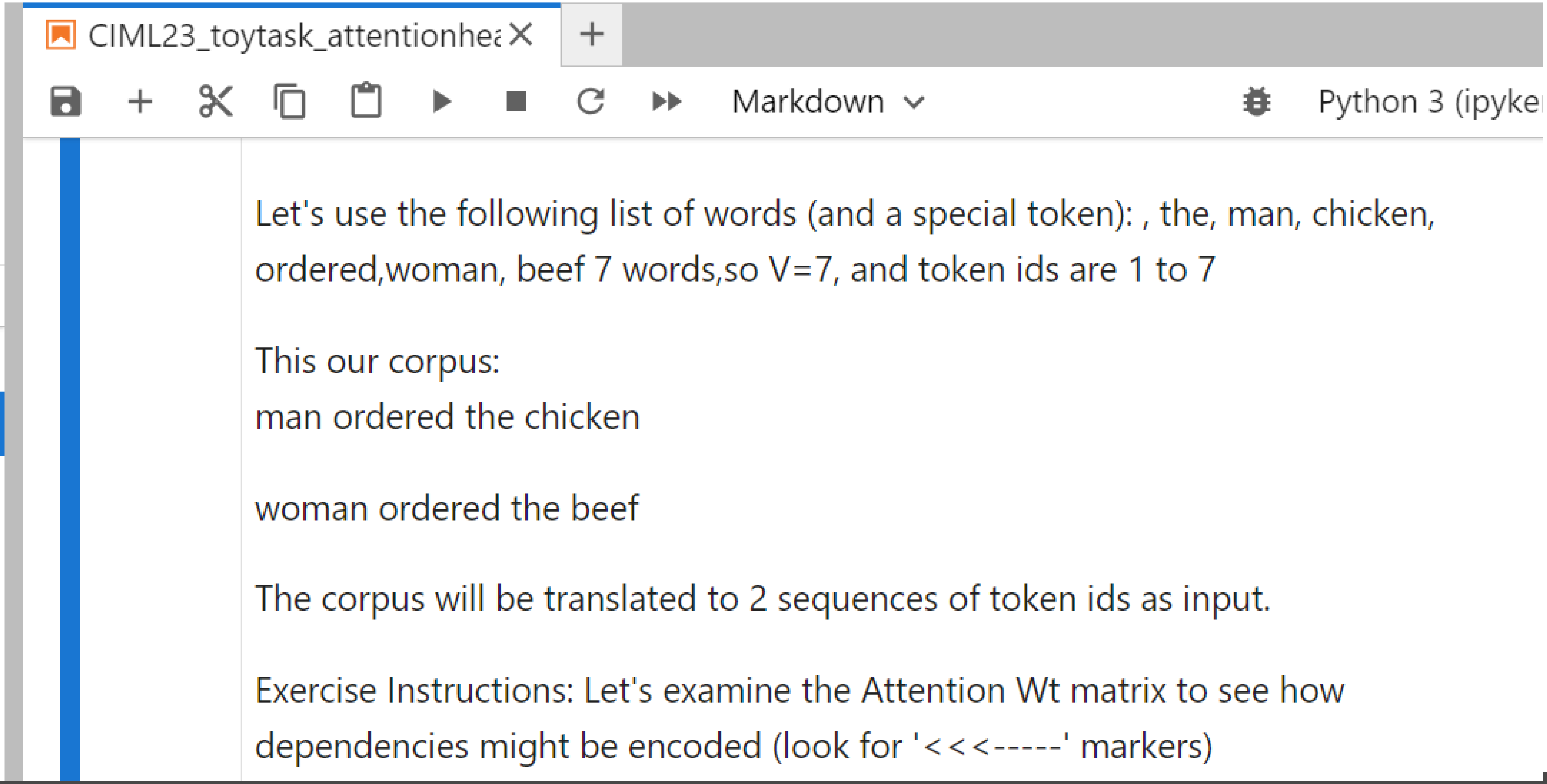
11 Layers/Functions

```
#Now build model to learn transformation for Q,K,V matrices

Xsequence      = tf.keras.layers.Input(shape=(T,V)) #the batch size is left unspecified
Pos_Input      = tf.keras.layers.Input(shape=(T)) #just the t=1...T integer
Pos_Embed      = tf.keras.layers.Embedding(T,V, input_length=T,name='PosEmbed')(Pos_Input) #input wi
Xinputs        = tf.keras.layers.Add()([Xsequence, Pos_Embed])

#now feed to Q,K,V transformations
Qmat        = tf.keras.layers.Dense(H,activation='linear',use_bias=False,name='Qmat')(Xinputs) #so f
Kmat        = tf.keras.layers.Dense(H,activation='linear',use_bias=False,name='Kmat')(Xinputs)
Vmat        = tf.keras.layers.Dense(H,activation='linear',use_bias=False,name='Vmat')(Xinputs)

#now apply QtoK take softmax, scale it , apply to V
QK           = tf.keras.layers.Dot(axes=(2))([Qmat,Kmat])  #it will treat each Batch item separately
QKscaled     = tf.keras.layers.Lambda(lambda x: x * scale_constant)(QK)    #for each x in QK mult by
Attn_Wts      = tf.keras.layers.Softmax(axis=2,name='AttnWts')(QKscaled, mask=Msk)        #apply mas
Vout        = tf.keras.layers.Dot(axes=1,name='Voutput')([Attn_Wts,Vmat])

my_attn_model   = tf.keras.Model(inputs = [Xsequence,Pos_Input], outputs=Vout)
```

Also, scale attn matrix by H

$$\underbrace{\mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)}V$$

An example of attention head with a toy task:
Run the “toytask_attention notebook” and observe the printed predictions and attention weights.   Try changing H – does it help/hurt?



Let's use the following list of words (and a special token): , the, man, chicken, ordered,woman, beef 7 words,so V=7, and token ids are 1 to 7

This our corpus:
man ordered the chicken

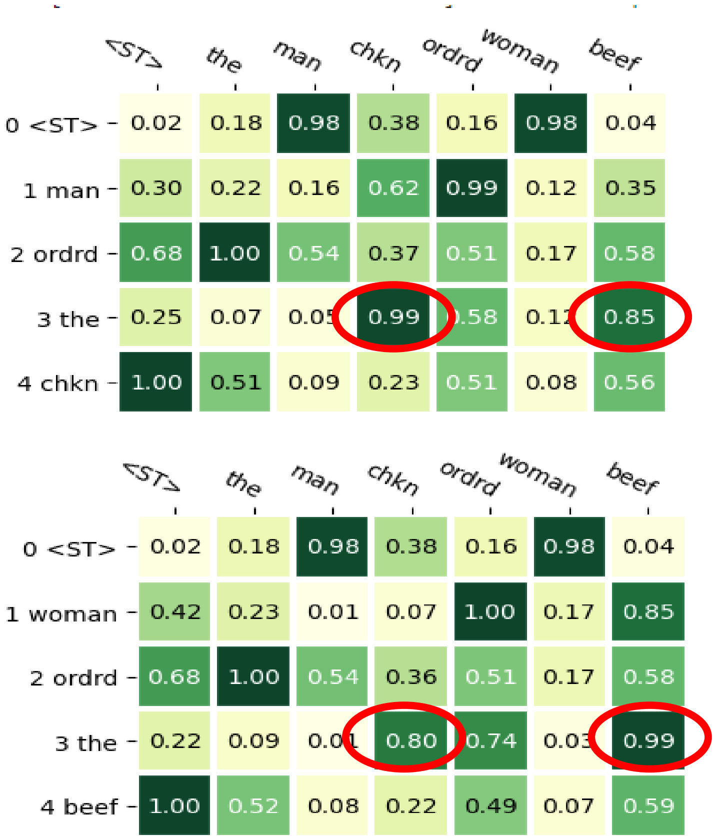
woman ordered the beef

The corpus will be translated to 2 sequences of token ids as input.

Exercise Instructions: Let's examine the Attention Wt matrix to see how dependencies might be encoded (look for '<<<-----' markers)

Output TxV predictions:

Notice that the→ [chicken or beef] predictions change depending on who's ordering



Output TxV predictions:

Notice that the→ [chicken or beef] predictions change depending on who's ordering

	<ST>	the	man	chkn	ordrd	woman	beef
0 <ST>	0.02	0.18	0.98	0.38	0.16	0.98	0.04
1 man	0.30	0.22	0.16	0.62	0.99	0.12	0.35
2 ordrd	0.68	1.00	0.54	0.37	0.51	0.17	0.58
3 the	0.25	0.07	0.05	0.99	0.58	0.12	0.85
4 chkn	1.00	0.51	0.09	0.23	0.51	0.08	0.56

	<ST>	the	man	chkn	ordrd	woman	beef
0 <ST>	0.02	0.18	0.98	0.38	0.16	0.98	0.04
1 woman	0.42	0.23	0.01	0.07	1.00	0.17	0.85
2 ordrd	0.68	1.00	0.54	0.36	0.51	0.17	0.58
3 the	0.22	0.09	0.01	0.80	0.74	0.03	0.99
4 beef	1.00	0.52	0.08	0.22	0.49	0.07	0.59

TxT Attn Wts:

Notice weights for "the" are different for 'man" or "woman"

	0 <ST>	1 man	2 ordrd	3 the	4 chkn
0 <ST>	1.00	0.00	0.00	0.00	0.00
1 man	0.06	0.94	0.00	0.00	0.00
2 ordrd	0.06	0.00	0.93	0.00	0.00
3 the	0.01	0.78	0.01	0.20	0.00
4 chkn	0.02	0.01	0.04	0.04	0.90

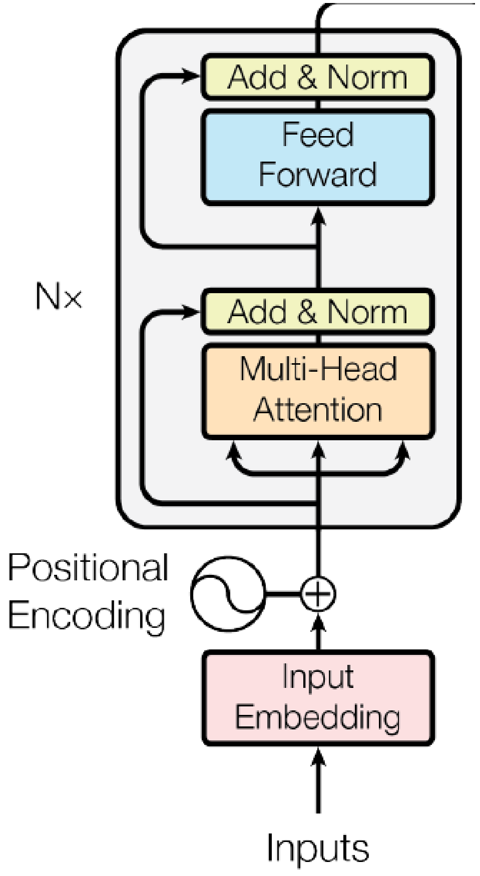
	0 <ST>	1 woman	2 ordrd	3 the	4 beef
0 <ST>	1.00	0.00	0.00	0.00	0.00
1 woman	0.10	0.90	0.00	0.00	0.00
2 ordrd	0.06	0.00	0.93	0.00	0.00
3 the	0.01	0.87	0.01	0.12	0.00
4 beef	0.01	0.02	0.04	0.04	0.89

- **pause**

Finally, a transformer

Include skip-add connections

Include Layer Normalization or DropOut layers

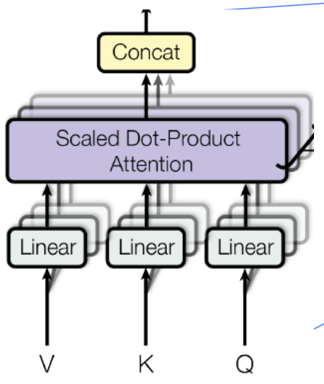


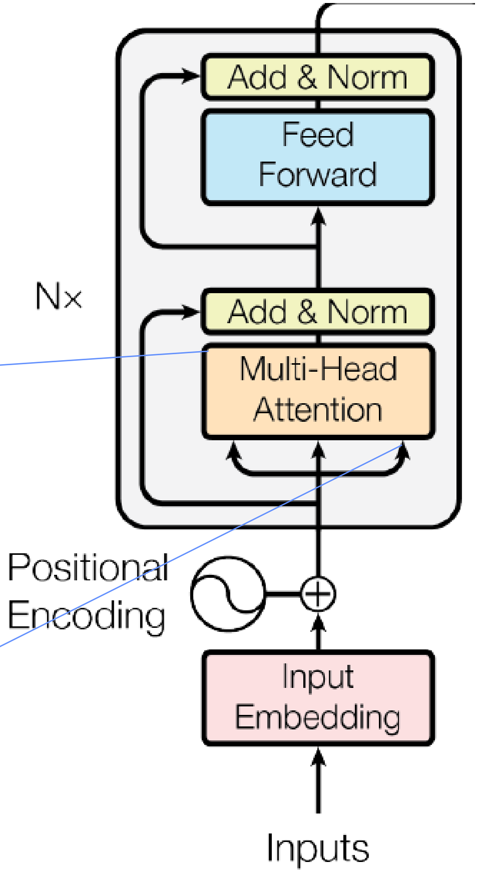
Finally, a transformer

Include skip-add connections

Include Layer Normalization or DropOut layers

Multi-Head – for N heads produce *Tx(H/N)* each



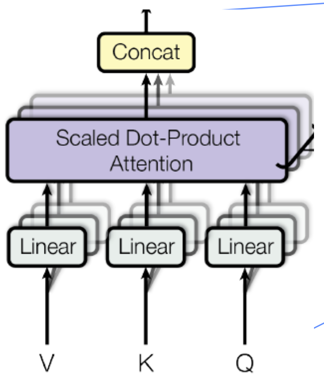


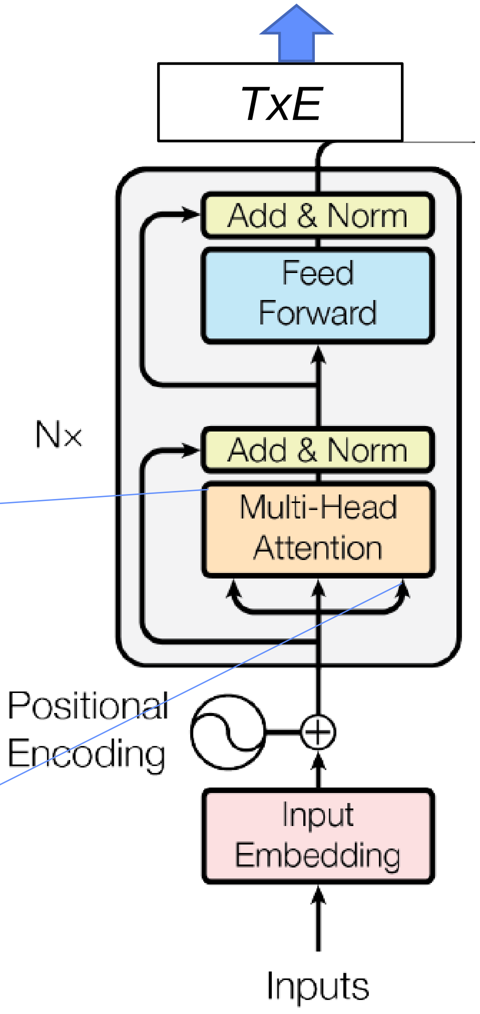
Finally, a transformer

Include skip-add connections

Include Layer Normalization or DropOut layers

Multi-Head – for N heads produce *Tx(H/N)* each





Add MLP layers on top –

output another *TxE* matrix

stackable!

Finally, a transformer

Include skip-add connections

Include Layer Normalization or DropOut layers

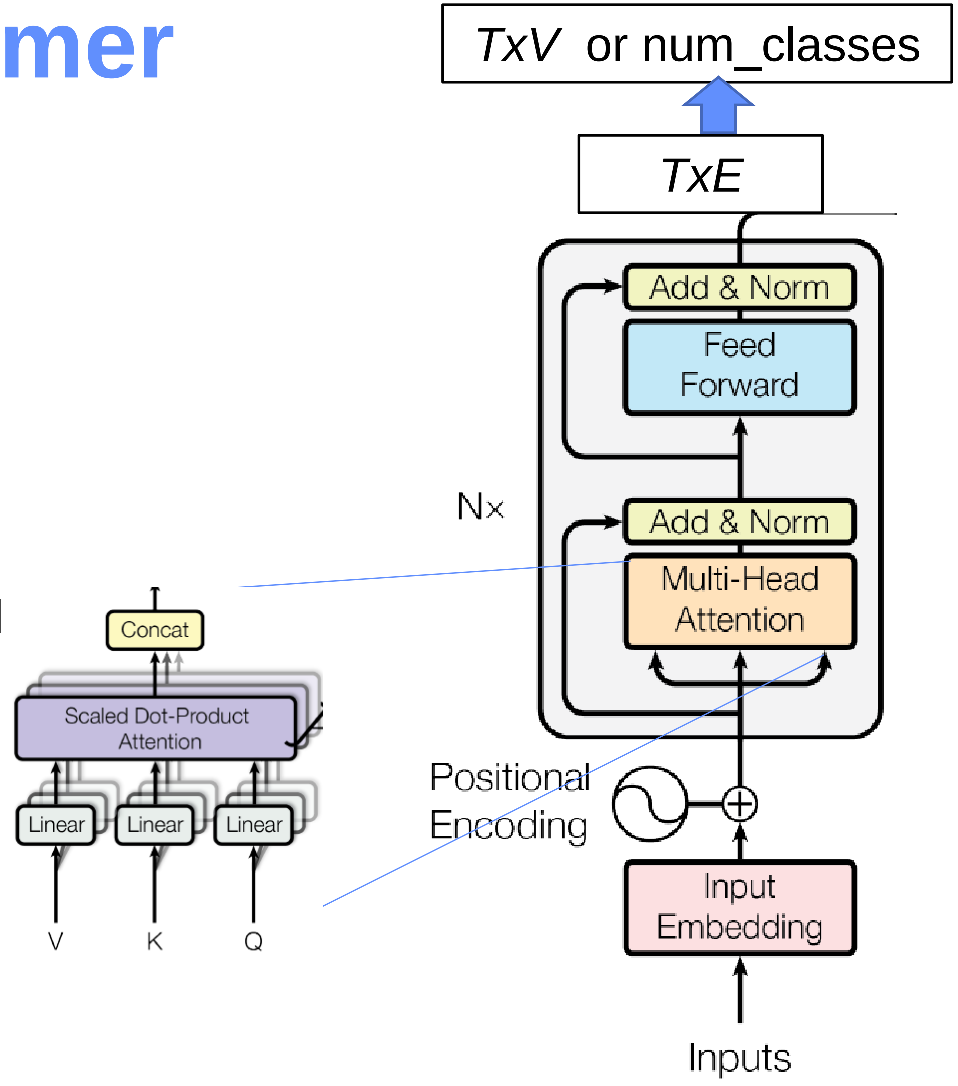
Multi-Head – for N heads produce *Tx(H/N)* each

Add MLP layers on top –

output another *TxE* matrix

or output final probabilities

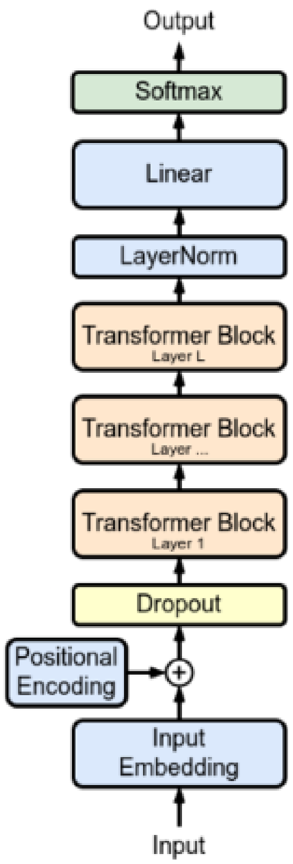
stackable!



2 kinds of training strategies

GPT – predict next word only look back at prior context (which could be a whole document)

Put mask on attention weights so that predictions only depend on previous tokens



BERT – *No attention mask* so all token dependencies can influence all other tokens predictions

Special tokens help create a variety of tasks

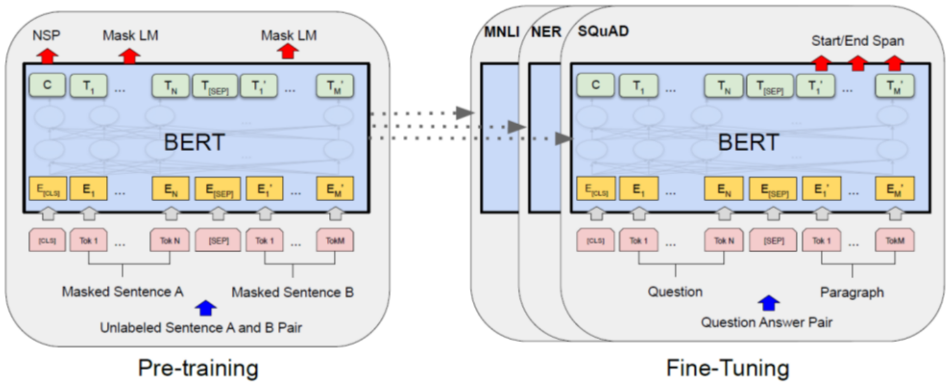
BERT (Bidirectional Encoder Representations from Transformers)

Goal: Train a model to develop general token-level AND sentence-level encoding

1 Pretrain on:

- fill-in-the-blank

- binary classification if 2 sentences go together



Devlin, etal, 2019

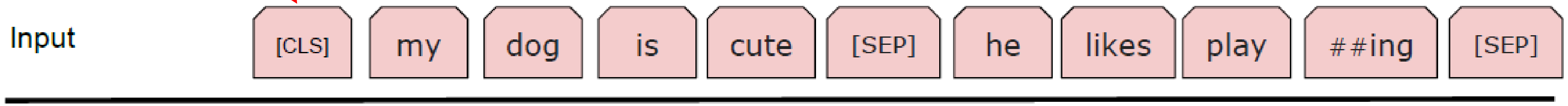
2 Fine tune on variety of tasks

BERT Input: 2 sentences

[CLS]
is'dummy' token
and always
present

[SEP] marks separation
between sentences or
sequences

[PAD] tokens help
fill out sequence to T
items



Devlin, etal, 2019

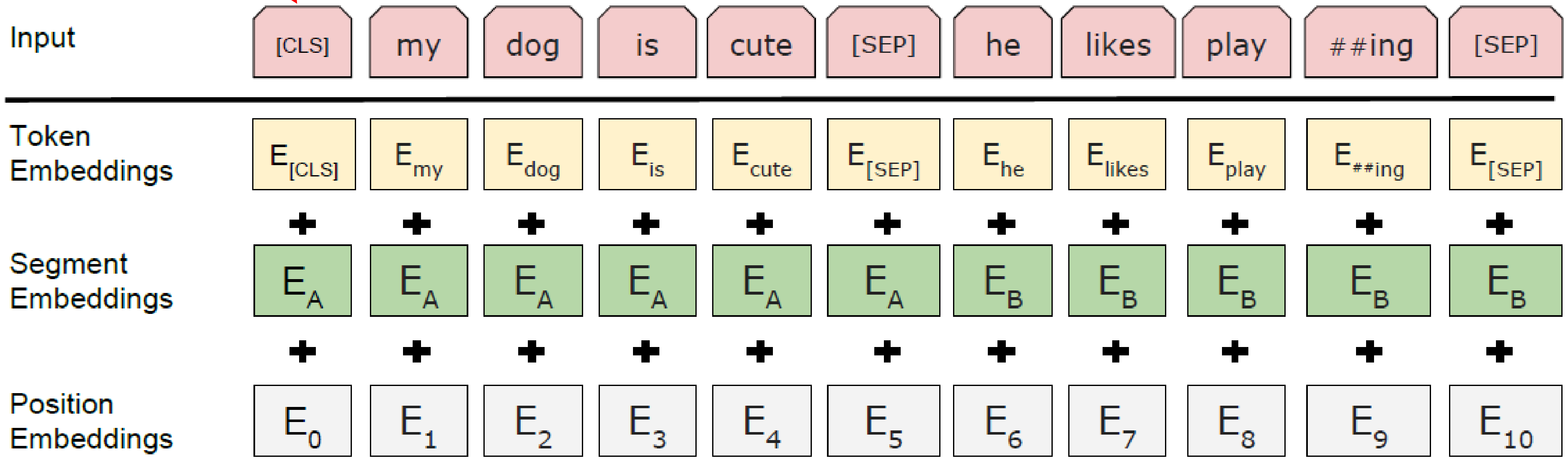
BERT Input: 2 sentences

[CLS] is'dummy' token and always present

[SEP] marks separation between sentences or sequences

[PAD] tokens help fill out sequence to T items

Add a TxE matrix for 'A" vs 'B' sentence



GPT (generative pre-trained transformer)

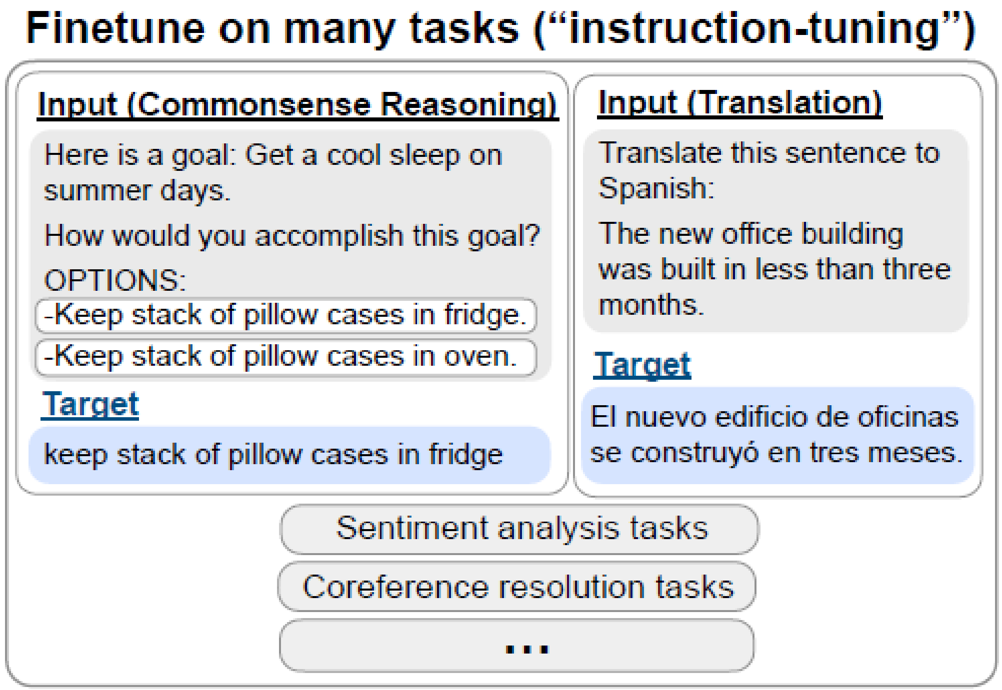
Goal: Train a transformer model at large scale so that it develops very general representations that are useful for many language tasks.

‘GPT3 shows strong performance on many NLP tasks so that with a few examples it nearly matches fine-tuned systems’

Lang. Models are Few Shot Learners
Brown, etal, 2020, openAI,

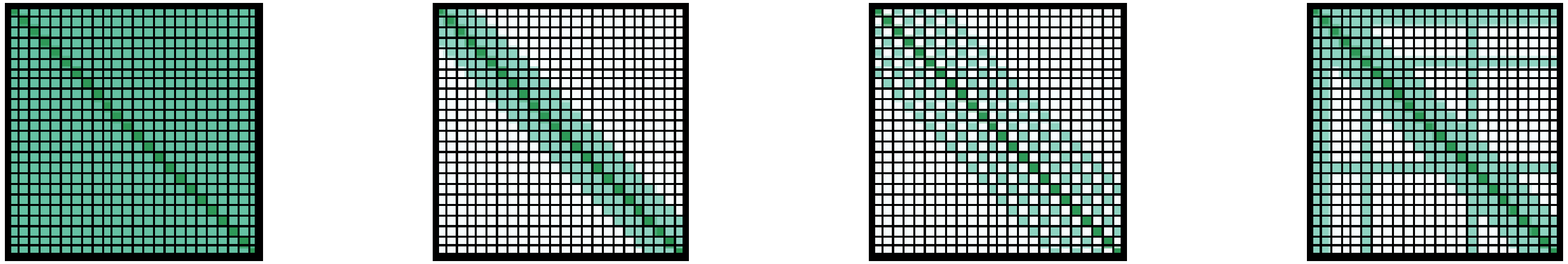
‘Instruction tuning’ improves models so they don’t need examples

Finetuned language models are zero-shot Learners. Wei et al, 2022, Google



Other strategies: Sparse Attention

Mistral - long sequence sparse, large attention matrix for ‘Longformer’ or ‘Sparse Transformer’



(a) Full n^2 attention   (b) Sliding window attention   (c) Dilated sliding window   (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

Generating Long Sequences with Sparse Transformers, Child etal, 2019
Longformer: The Long-Document Transformer Beltagy etal. Allen Institute, 2020

- **Transformers for Science applications**

Can anything be cast as a kind of sentence, or an arrangement of tokens?

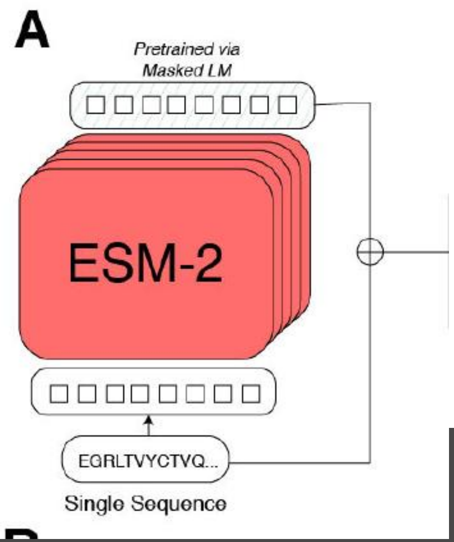
ESM Fold model

Language models of protein sequences at the scale of evolution enable accurate structure prediction

Lin etal, Meta Research 2022

Atom level structure prediction

Uses protein sequence as input to transformer layers (like LLM)



ESM Fold model

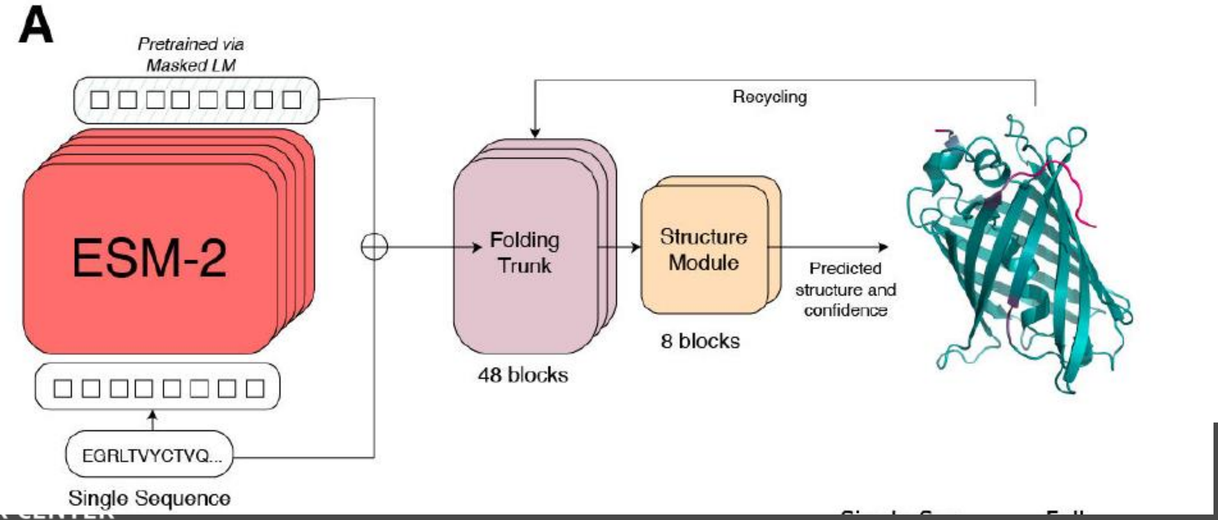
Language models of protein sequences at the scale of evolution enable accurate structure prediction

Lin etal, Meta Research 2022

Atom level structure prediction

Uses protein sequence as input to transformer layers (like LLM)

Predicts a map of protein contact which gets *iteratively refined* by a ‘folding block’ transformer and structure module (similar to AlphaFold2, but faster)



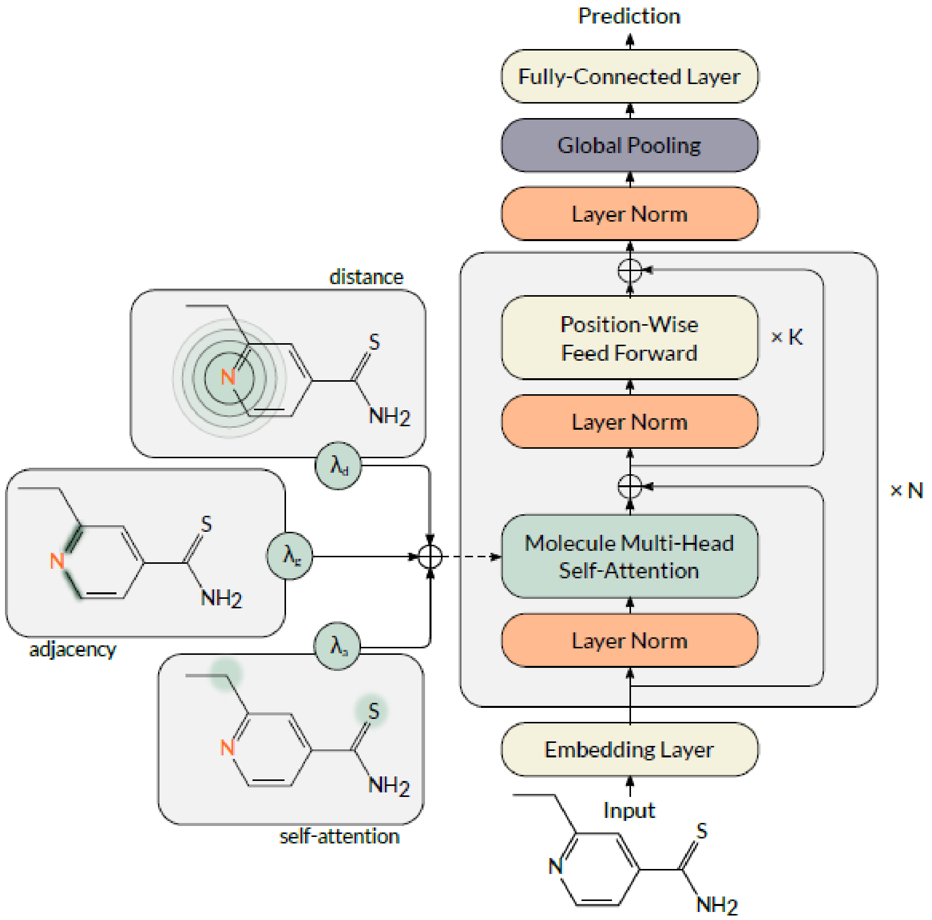
Molecule Attention Transformer

(Maziarka et al. 2020)

Molecular property prediction

Uses the set of atoms as input (like sentence tokens)

Includes spatial information by using a sum of the attention matrix, a distance matrix, and an adjacency matrix.



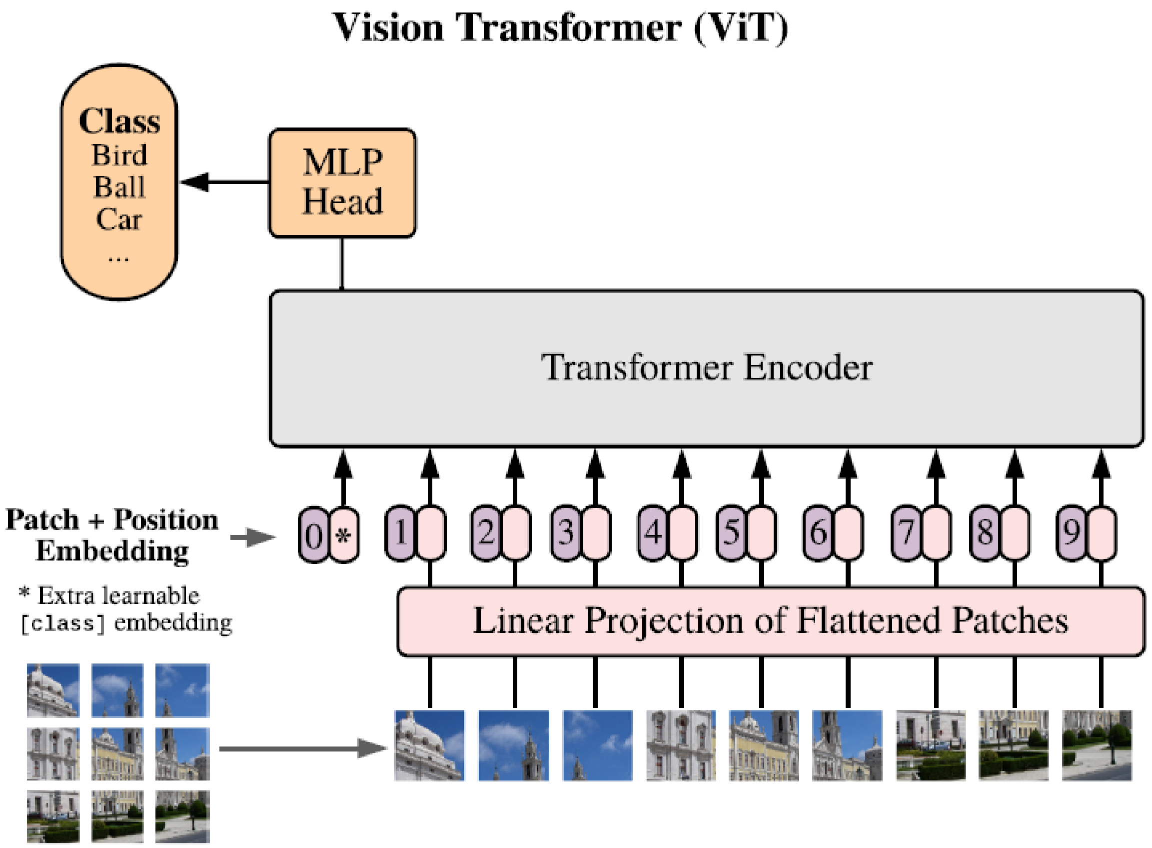
The Visual Transformer (ViT)

An image is worth 16x16 words:
Transformers for image recognition at scale
Adosovitski, et al, 2021, Google Research

Uses a sequences of image patches (16x16) like a sentence of tokens (ie 224x224 pixels is 16x16 patches of 14x14 pixels)

Uses a classification token like Bert to learn image output classes

Competitive or better than CNNs but might need more data

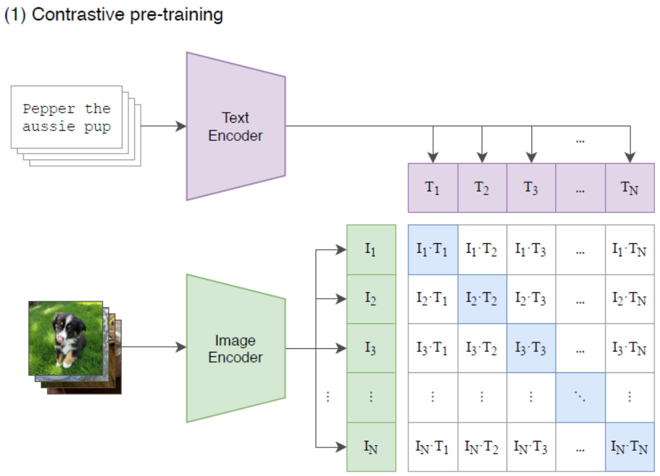


- **Combining images and text often makes DL work better, or more generic, for image or text tasks**

CLIP – Contrastive Language-Image Pretraining

Learning Transferable Visual Models From Natural Language Supervision

Radford et al, 2021, Open AI



Uses 400M images and captions for training

Learns a multi-modal embedding

Maximizes embedding similarity of captions with it's image; minimizes embedding similarity of captions with other images

CLIP – Contrastive Language-Image Pretraining

Learning Transferable Visual Models From Natural Language Supervision

Radford et al, 2021, Open AI

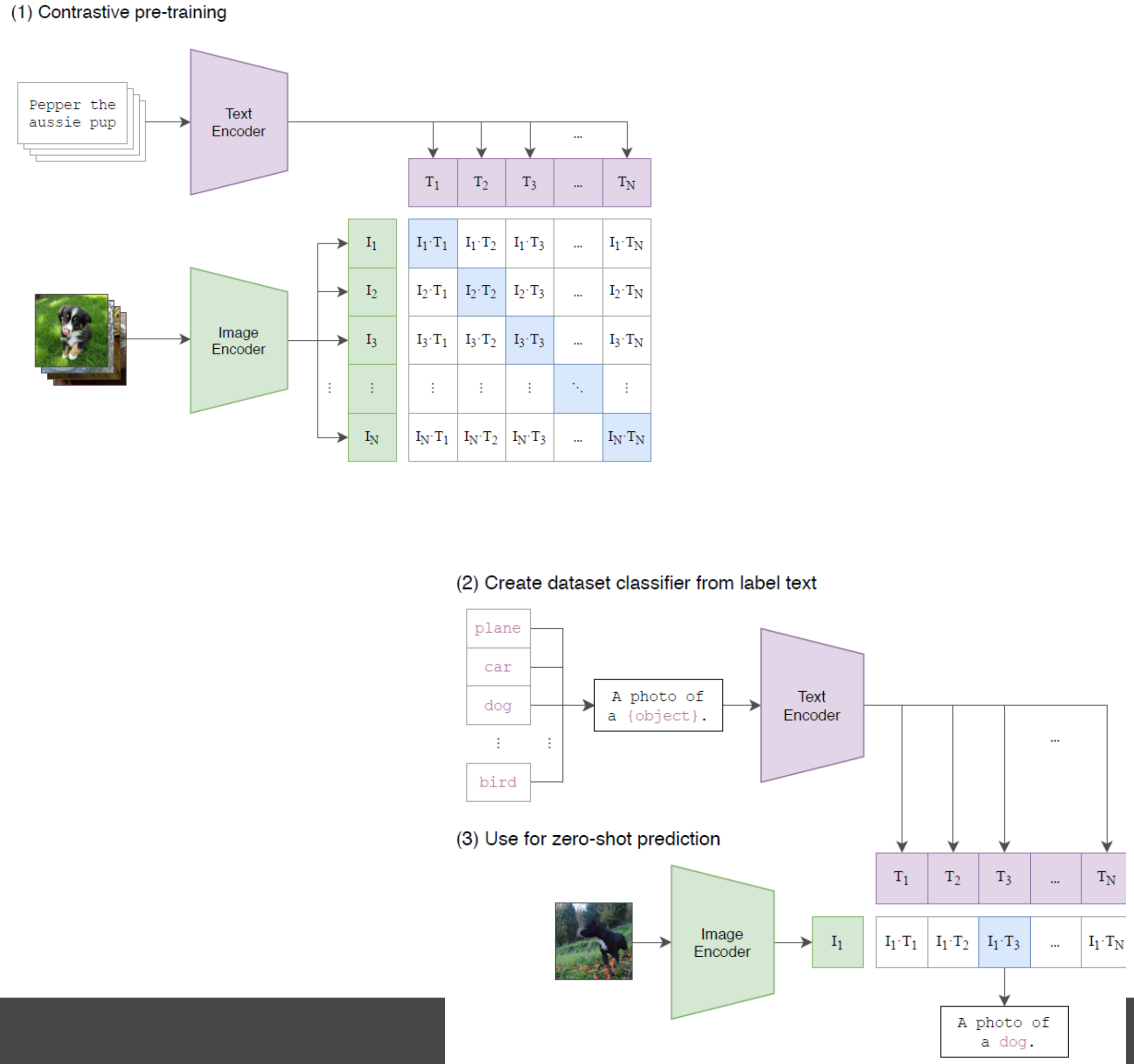
Uses 400M images and captions for training

Learns a multi-modal embedding

Maximizes embedding similarity of captions with it's image; minimizes embedding similarity of captions with other images

Performs classification by prompting it with an image and possible captions

Note: CLIP with diffusions gets close to DALL-E



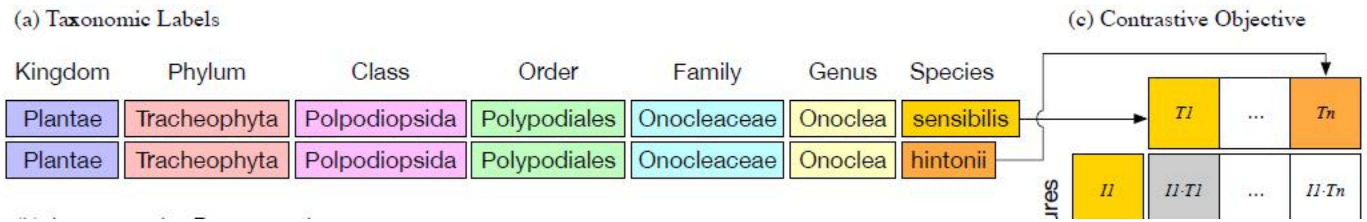
BIOCLIP: A Vision Foundation Model for the Tree of Life

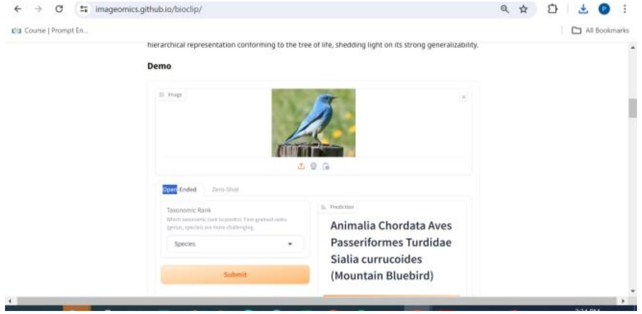
Stevens, etal 2024 OSU

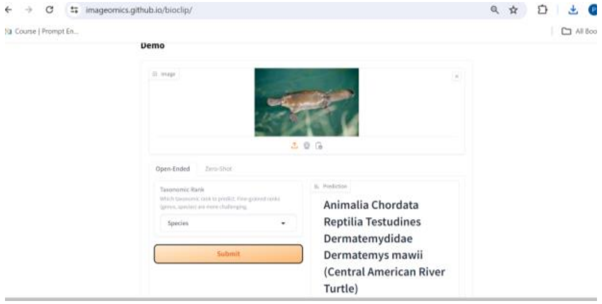
Uses pre-trained CLIP for a base

Uses Tree-of-Life 10M dataset of biology images with taxonomic labels

The taxonomic hierarchy is presented as a sequence of words for different species







BIOCLIP: A Vision Foundation Model for the Tree of Life

The model has github repo site and can be installed on Expanse directly as follows:

(see bioclip commands text file)

1. Get into interactive session on Expanse node, start singularity shell
2. pip install from github to a user folder
3. Export PYTHONPATH
4. Point program to your images and run

```
#1 Request a GPU node
#use jupyter alias for gpushared.. but don't get into notebook

#2 ssh into that node
squeue -u $USER
ssh exp-X-X

#Note: X-X should be the expanse node id numbers

#3 Load modules

module load gpu
module load slurm
module load singularitypro/3.11

#4 Run singularity shell  command

singularity shell --nv /cm/shared/apps/containers/singularity/py
```

```
Singularity> pip install git+https://github.com/Imageomics/pybiocl
```

```
Singularity> export PYTHONPATH=/home/$USER/Local_BioClip/local/lib/p
Singularity> echo $PYTHONPATH
/home/p4rodrig/Local_BioClip/local/lib/python3.10/dist-packages/
```

```
Singularity> python3 run_bc.py
open_clip_pytorch_model.bin: 100%|                    |
open_clip_config.json: 100%|
txt_emb_species.npy: 100%|                            |
txt_emb_species.json: 100%|                          | 65
Sialia currucoides - 0.9975905418395996
Tersina viridis - 0.0009066067868843675
Eumyias thalassinus - 0.00020906853023916483
Coracias garrulus - 0.00014162158186081797
Gymnorhinus cyanocephalus - 0.00013305182801559567
Singularity>
```

end