Deep Learning Topics: Special Connections Part IITransformers

Paul Rodriguez, PhD

(SDSC)

June 2025

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 …

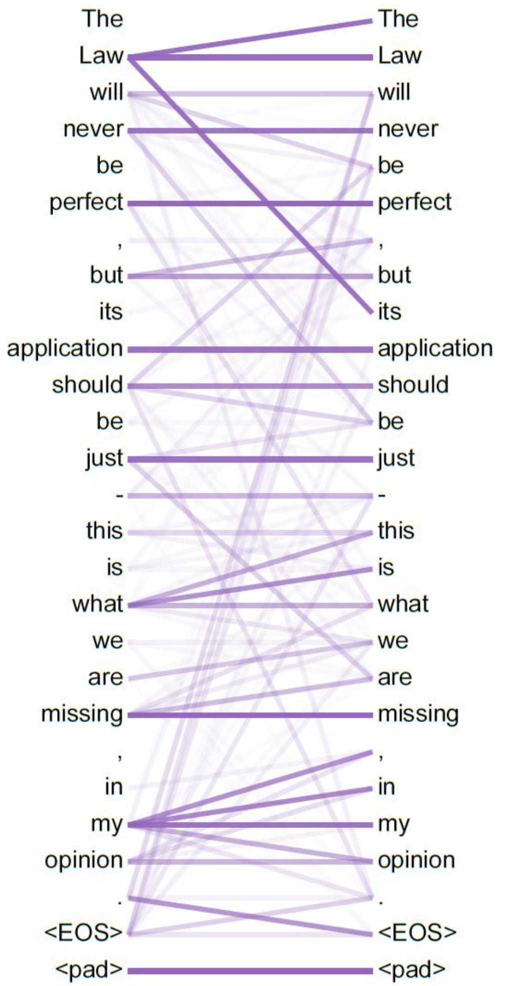
Dependences of Language

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 …



In language, there are many dependencies and interactions between words

A toy problem to get some intuition

- Let's use the following list of 7 tokens (ie vocabulary of 7words)

  1 <**s**tart>, 2 the, 3 man, 4 chicken, 5 ordered, 6 woman, 7 beef

- For input data, let's have these 2 sentences and token id sequences:

  <start> man ordered the chicken   [1,3,5,2,4]

- <start> woman ordered the beef   [1,6,5,2,7]

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

The toy task: predict next word

Predict the next word for the first input sentence after each word:

<Start> man ordered the chicken

The toy task: predict next word

Predict the next word for the first input sentence after each word:

<Start> man ordered the chicken

A basic solution is a bigram matrix:
X=Sequence-to-Word, size is TxV

Next Token (ie word) Prediction

Token Sequence

Pos	Word	<strt>	The	Man	Chik	Ord	Wmn	Beef
1	<start>			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

Predict the next word for the first input sentence after each word:

<Start> man ordered the chicken

A basic solution is a bigram matrix:

X=Sequence-to-Word, size is TxV

Next Token (ie word) Prediction

Token Sequence

Pos	Word	<strt>	The	Man	Chik	Ord	Wmn	Beef
1	<start>			0.5			0.5	
2	Man					1.0		
3	Orde.r		1.0					
4	The				0.5			0.5
5	Chick.	1.0						

The bigram doesn't use context

The toy task: predict next word

Predict the next word for the first input sentence after each word:

<Start> man ordered the chicken

A basic solution is a bigram matrix:
X=Sequence-to-Word, size is TxV

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

After ‘Man’ the → chicken = 1.0

After ‘Woman’ the → beef = 1.0

nce

Next Token (ie word) Prediction

Pos	Word	<strt>	The	Man	Chik	Ord	Wmn	Beef
1	<start>			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

Predict the next word for the first input sentence after each word:

<Start> man ordered the chicken

A basic solution is a bigram matrix:
X=Sequence-to-Word, size is TxV

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

After 'Man' the → chicken = 1.0

After 'Woman' the → beef = 1.0

nce

Next Token (ie word) Prediction

Pos	Word	<strt>	The	Man	Chik	Ord	Wmn	Beef
1	<start>			0.5			0.5	
2	Man					1.0		
3	Orde.r		1.0					
4	The				0.5			0.5
5	Chick.	1.0						

We want 'man' context to pass information

The attention idea

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

E.G. for *X a TxV* matrix of possible predictions, we want to transform *X into contextualized predictions*

X= Sequence-to-Word is TxV

$$\begin{pmatrix} 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0.5 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow$$

Contextualized Predictions

The attention idea

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

E.G. for *X a TxV* matrix of possible predictions, we want to transform *X into contextualized predictions*

We need a W a TxT matrix – aka 'attention' weights that has word dependencies

X= Sequence-to-Word is TxV

$$\begin{pmatrix} 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0.5 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow$$

Contextualized Predictions

The attention idea

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

E.G. for *X a TxV* matrix of possible predictions, we want to transform *X into contextualized predictions*

We need a W a TxT matrix – aka 'attention' weights that has word dependencies

X= Sequence-to-Word is TxV **Contextualized Predictions**

$$W_{TxT} * \begin{pmatrix} 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0.5 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The attention idea

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

E.G. for *X a TxV* matrix of possible predictions, we want to transform *X into contextualized predictions*

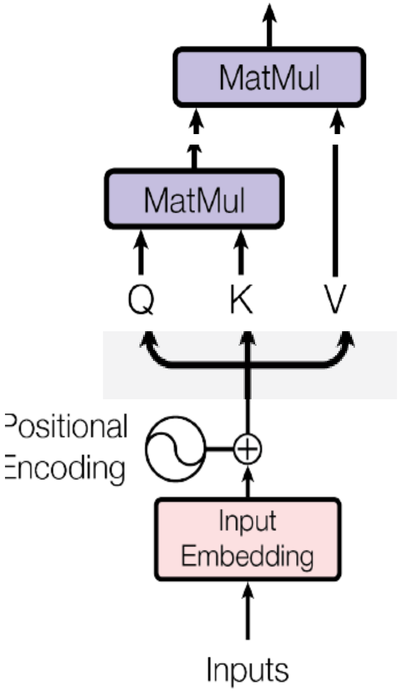
We need a W a TxT matrix – aka 'attention' weights that has word dependencies

X= Sequence-to-Word is TxV **Contextualized Predictions**

$$W_{TxT} * \begin{pmatrix} 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0.5 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

W should reflect the interdependencies of the words in the input
Q: Where should W come from? (Hint: what does all AI need)

- **Let's build up the attention architecture**



Get input embeddings

First, get sequence of token embeddings using 1-hot input to E units – see next slide

[1,3,5,2,4] → X_{TxE}

Sidenote: embedding (encoding) for words (ie tokens)

Each word (or word part) is assigned a ‘token id’ 1 to V=50K

Let X be input vector with only one value equal to 1 (a 1-hot vector) determined by token id

X is Vx1

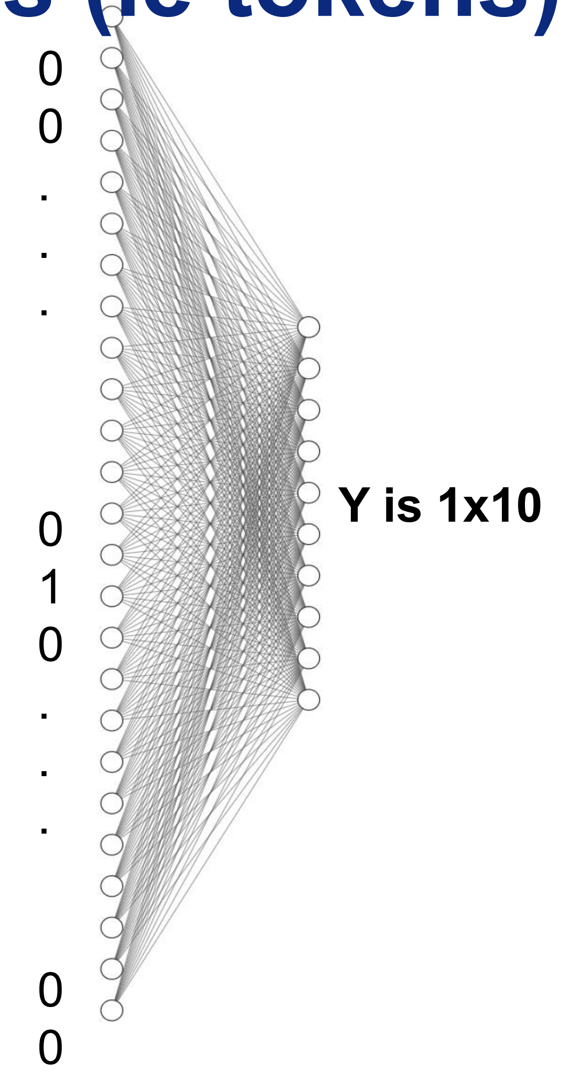
0
0
.
.
.
0
1
0
.
.
.
0
0

Sidenote: embedding (encoding) for words (ie tokens)

Each word (or word part) is assigned a ‘token id’ 1 to V=50K

Let X be input vector with only one value equal to 1 (a 1-hot vector) determined by token id

Use a one layer network to make an embedding

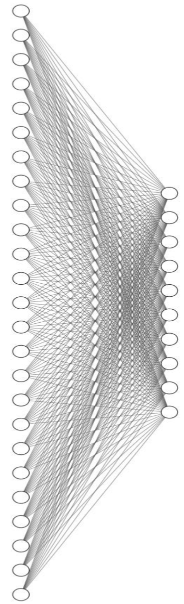


In this picture, each token id is converted to a lower dimensional vector with size 1x10

Each token sequence of T vectors is Tx10

(back to) Get input embeddings

First, get sequence of token embeddings using 1-hot input to E units



[1,3,5,2,4] → X_{TxE}

Get input and add in position info

First, get sequence of token embeddings using 1-hot input to E units

[1,3,5,2,4] → X_{TxE}

Then do the same for positions 1…T

[1,2,3,4,5] → P_{TxE}

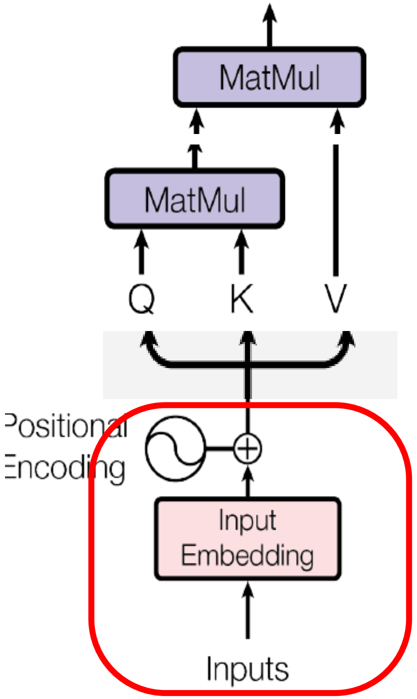
Get input and add in position info

First, get sequence of token embeddings using 1-hot input to E units

[1,3,5,2,4] → X_{TxE}

Then do the same for positions 1…T

[1,2,3,4,5] → P_{TxE}



X + P is final TxE matrix of input embeddings

Embeddings for attention weights

Take Input Embeddings and build a ‘Query’ (Q) and ‘Key’ (K) embedding matrix *of size TxH*

$$X + P \rightarrow Q_{TxH}$$

$$X + P \rightarrow K_{TxH}$$

The idea is to make new embeddings that can useful for word-to-word dependencies

Embeddings for attention weights

Take Input Embeddings and build a ‘Query’ (Q) and ‘Key’ (K) embedding matrix *of size TxH*

$X + P \rightarrow Q_{TxH}$

$X + P \rightarrow K_{TxH}$

The idea is to make new embeddings that can useful for word-to-word dependencies

Now let $W = Q{*}K'$

Notice that every token’s embedding gets to ‘interact’ with every other token’s embedding to make up the *TxT* elements of ***W***

Embeddings for attention weights

Take Input Embeddings and build a 'Query' (Q) and 'Key' (K) embedding matrix *of size TxH*

$X + P \rightarrow \quad Q_{TxH}$

$X + P \rightarrow \quad K_{TxH}$

The idea is to make new embeddings that can useful for word-to-word dependencies

Now let $W = Q*K'$

Notice that every token's embedding gets to 'interact' with every other token's embedding to make up the *TxT* elements of ***W***

$X + P \rightarrow \quad V_{TxV}$

Finally, instead of a pre-built bigram matrix, use another embedding for a 'Value' V matrix

Embeddings for attention weights

Take Input Embeddings and build a 'Query' (Q) and 'Key' (K) embedding matrix *of size TxH*

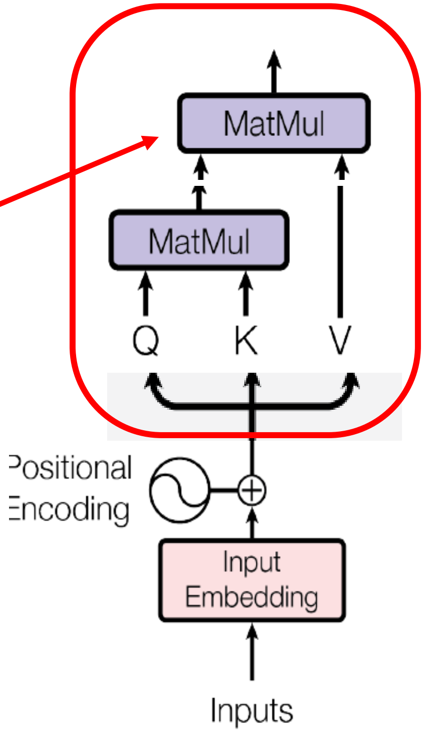
$X + P \rightarrow \; Q_{TxH}$

$X + P \rightarrow \; K_{TxH}$

Now, we can take sigmoid of W * V to get TxV predictions from 0 to 1

Now let $W = Q*K'$

$X + P \rightarrow \; V_{TxV}$



In class exercise:

An example of attention head with a toy task:

Run the “toytask_attention notebook” and observe the printed predictions and attention weights.

Markdown Python 3 (ipyke

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

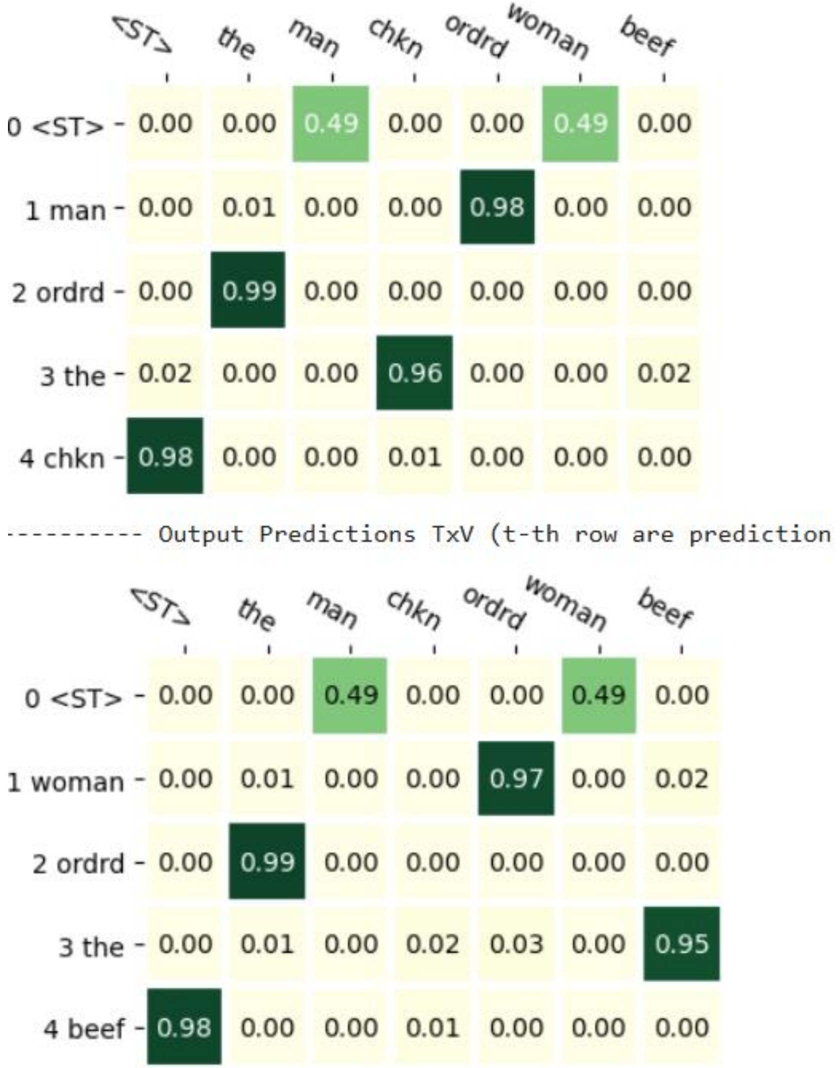
The corpus will be translated to 2 sequences of token ids as input.

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

Output TxV predictions

Notice <start> → [man or woman at 0.50]

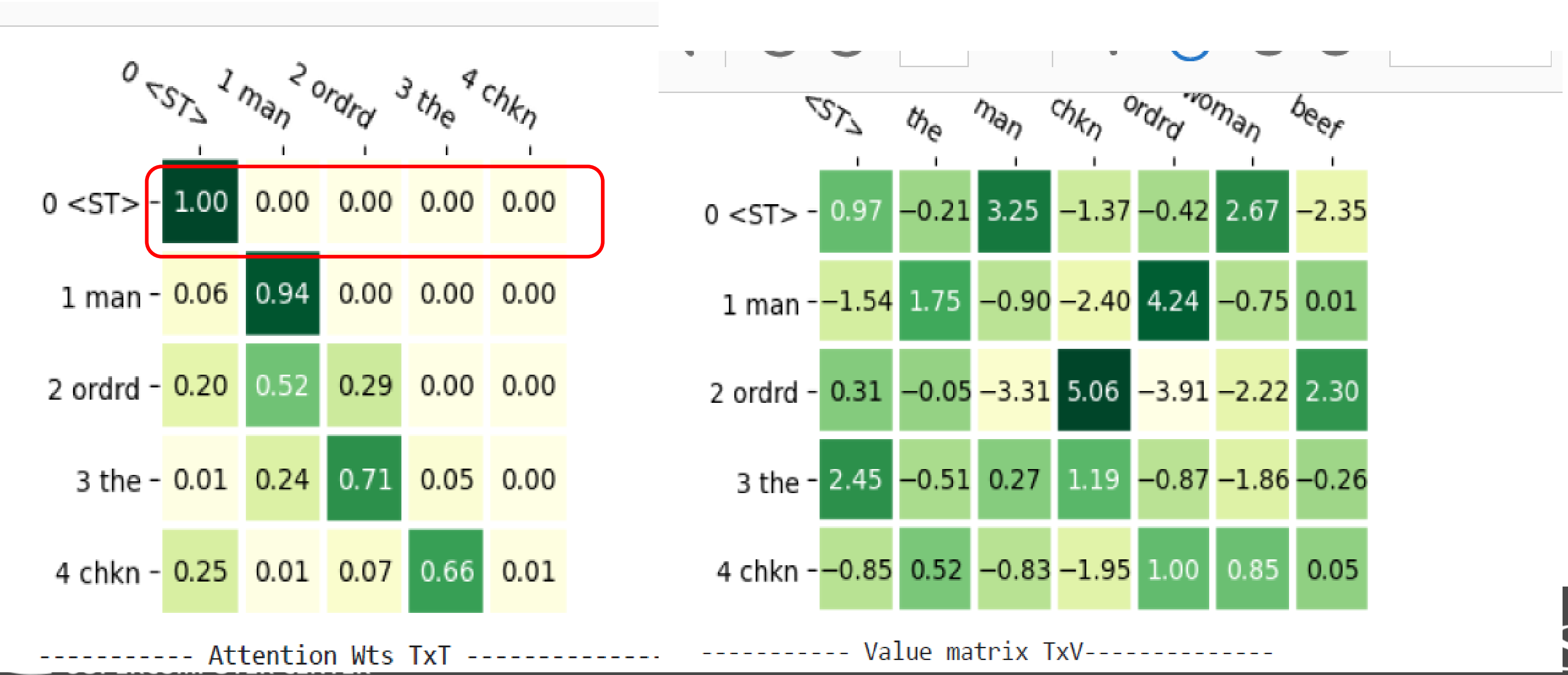
Notice that the→ [chicken or beef] predictions change depending on who's ordering



Here are the W (attention) and V (value) matrices for [<st> man ordrd the chckn]

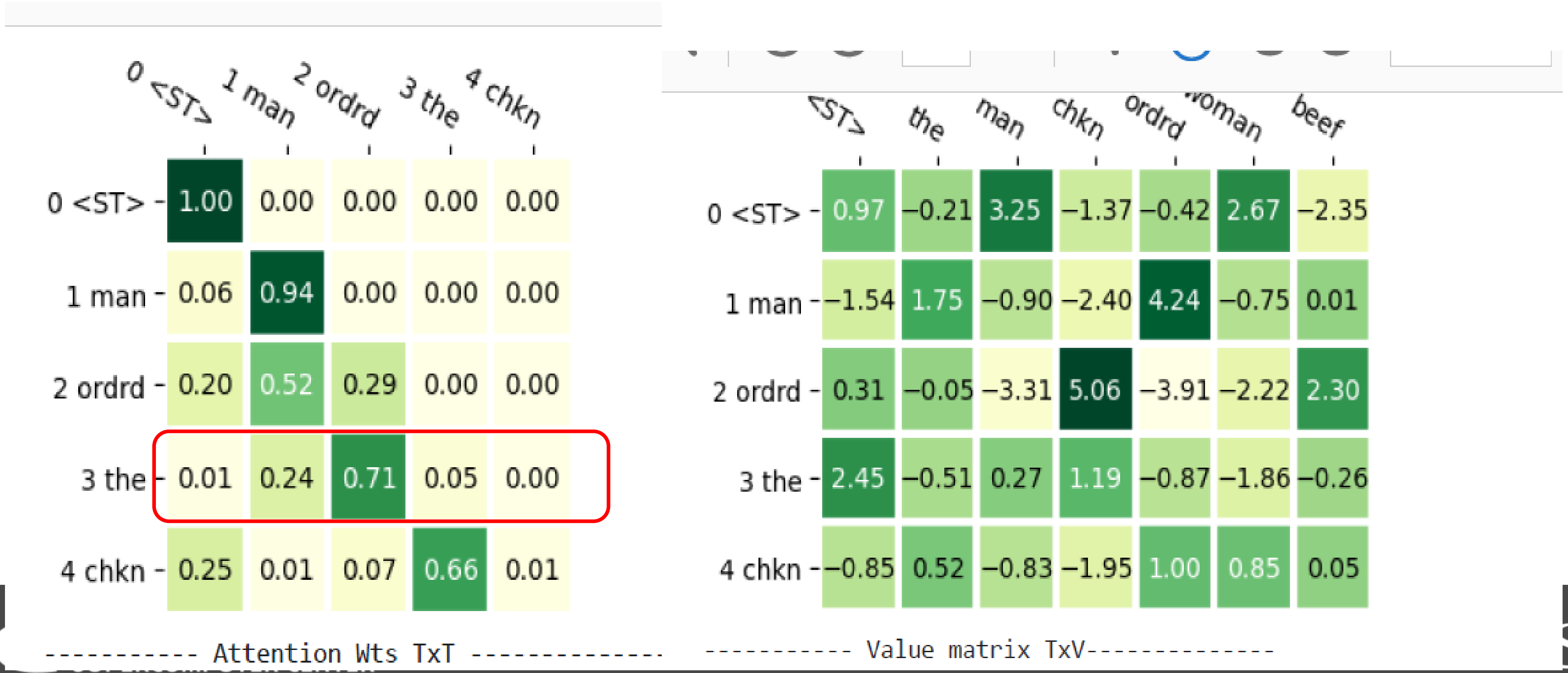
Notice the upper diagonal of W are masked so that predictions don't "look ahead"

Notice the <Start> token only depends on itself, and only picks off top row of V



Here are the W (attention) and V (value) matrices for [<st> man ordrd the chckn]

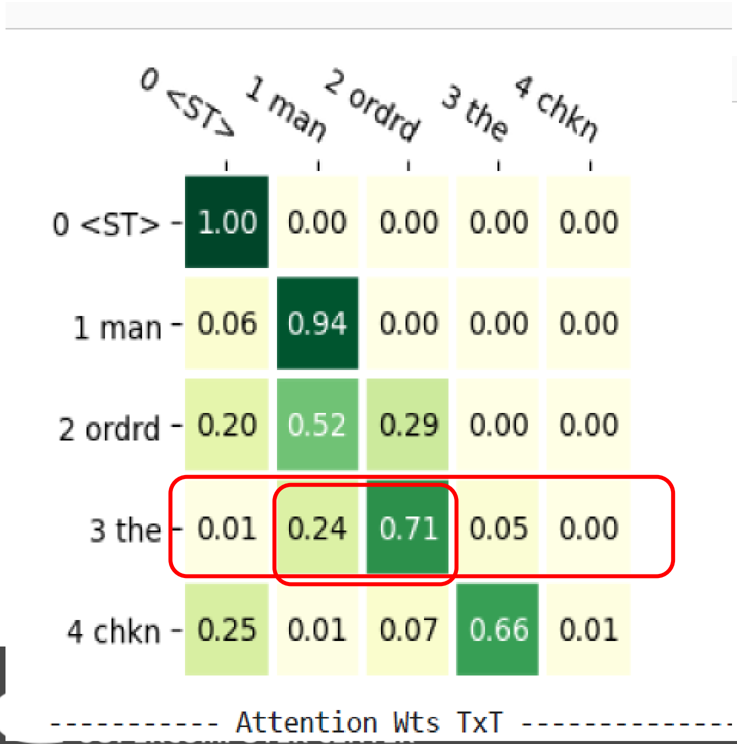
The 4th row of W 1xT times V makes the 1xV predictions of ‘the’

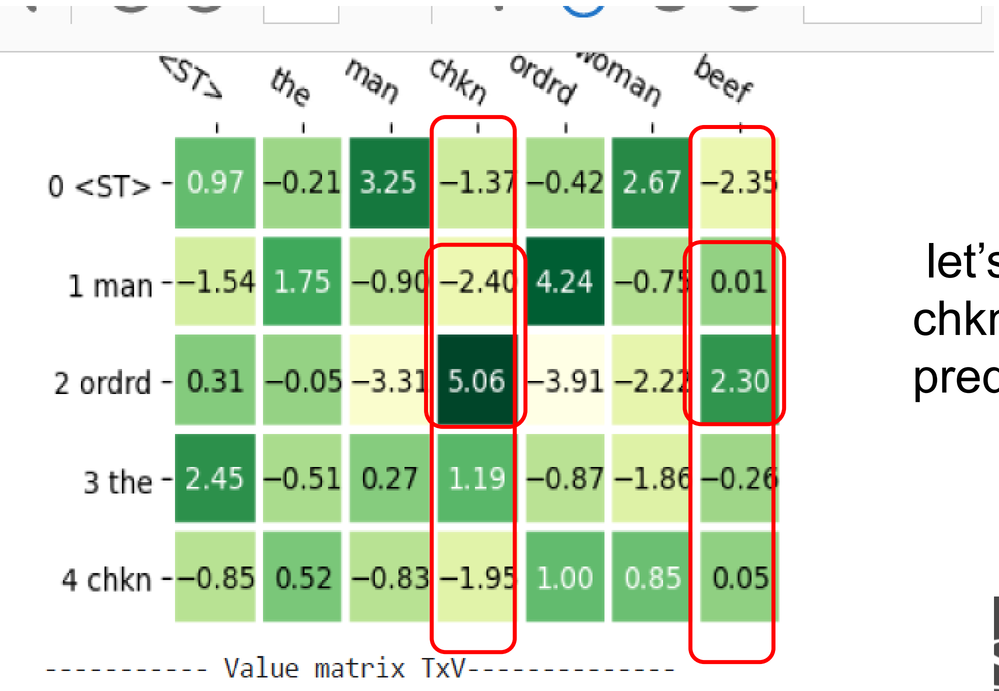


Here are the W (attention) and V (value) matrices for [<st> man ordrd the chckn]

The 4th row of W 1xT times V makes the 1xV predictions of ‘the’

only 2 values in 4th row > 0





let’s focus on chkn, beef predictions

Here are the W (attention) and V (value) matrices for [<st> **wmn** ordrd the **beef**]:

The 4th row of W 1xT times V makes the 1xV predictions of ‘the’

And let’s focus on ‘chkn’ vs ‘beef’

----------- Attention Wts TxT -------------

	0 <ST>	1 woman	2 ordrd	3 the	4 beef
0 <ST>	1.00	0.00	0.00	0.00	0.00
1 woman	0.01	0.99	0.00	0.00	0.00
2 ordrd	0.15	0.62	0.22	0.00	0.00
3 the	0.00	0.52	0.45	0.03	0.00
4 beef	0.26	0.00	0.07	0.66	0.01

only 2 values in 4th row > 0

----------- Value matrix TxV-------------

	<ST>	the	man	chkn	ordrd	woman	beef
0 <ST>	0.97	−0.21	3.25	−1.37	−0.42	2.67	−2.35
1 woman	−1.90	2.02	−1.95	−2.84	4.43	−1.68	0.97
2 ordrd	0.31	−0.05	−3.31	5.06	−3.91	−2.22	2.30
3 the	2.45	−0.51	0.27	1.19	−0.87	−1.86	−0.26
4 beef	−0.91	0.45	−0.79	−1.98	1.03	0.89	0.05

Pause –

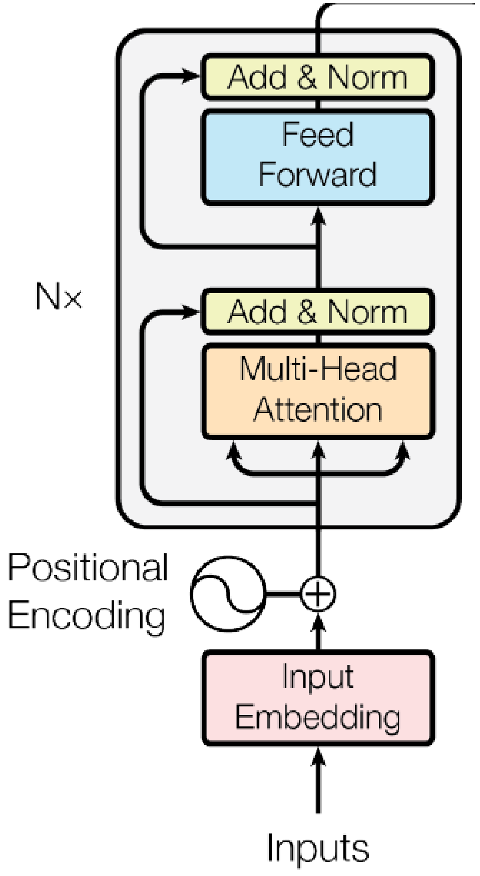
- **Consider what if the man & woman orders switched for different restaurants?**

- **Let scale up to transformers**

Finally, a transformer

Include skip-add connections

Include Layer Normalization or DropOut layers

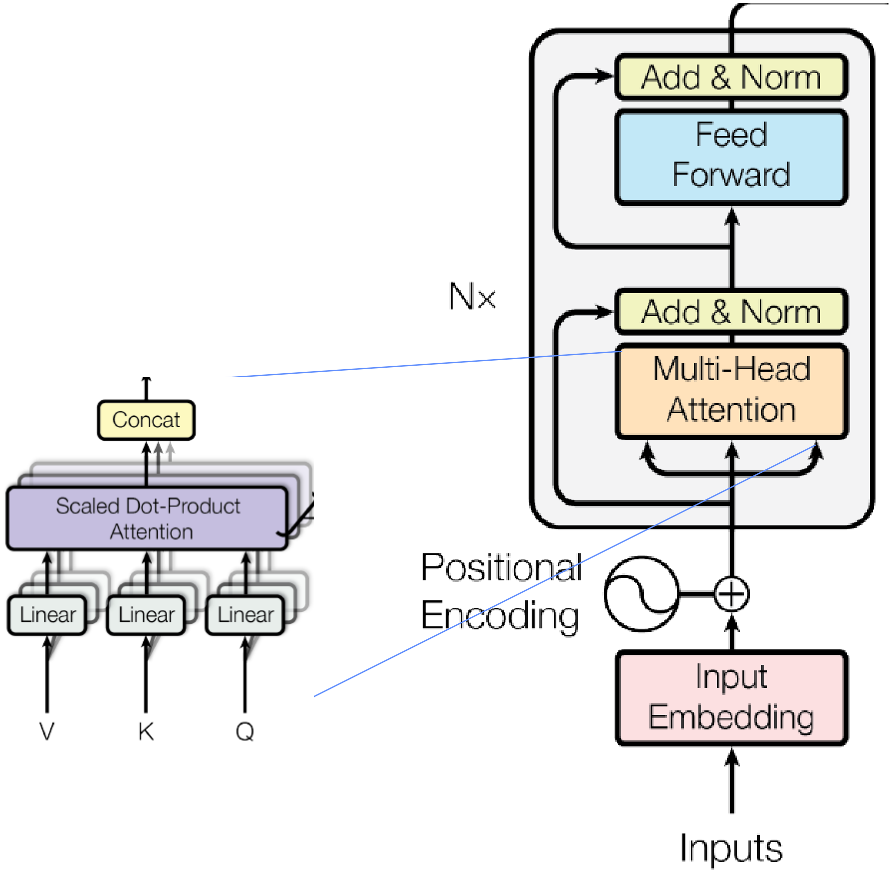


Finally, a transformer

Include skip-add connections

Include Layer Normalization or DropOut layers

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

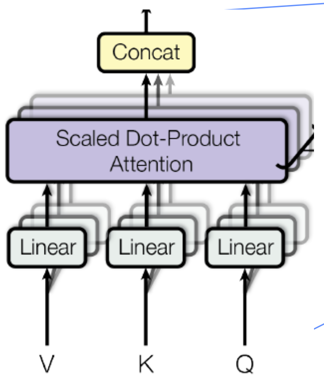


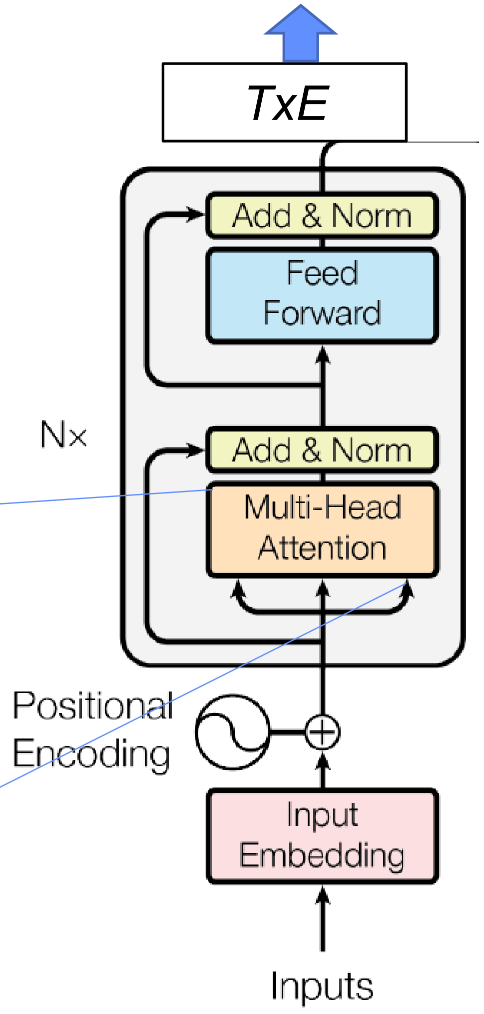
Finally, a transformer

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

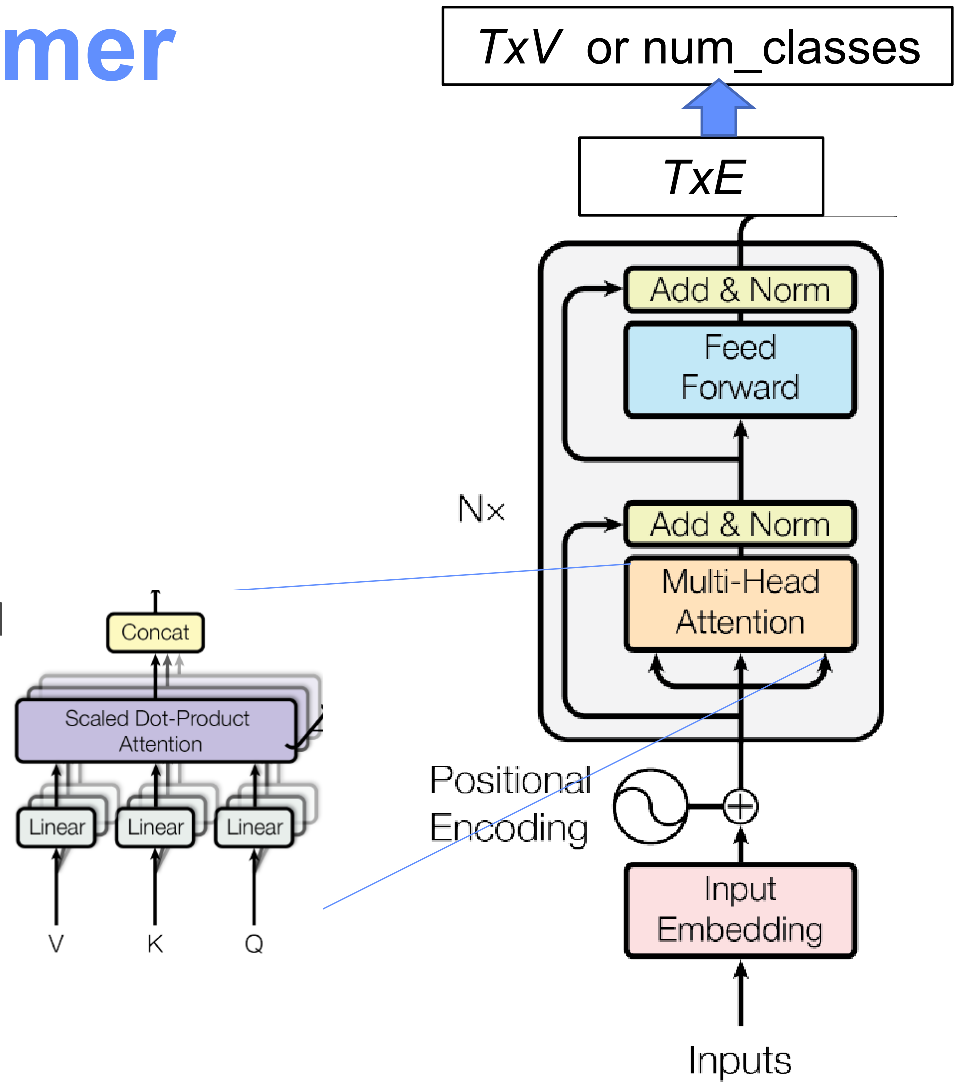
Multi-Head – for N heads produce *Tx(H/N)* each

Add MLP layers on top –

output another *TxE* matrix

or output final probabilities

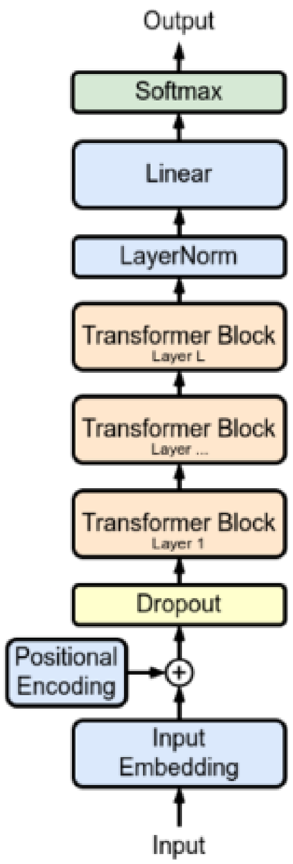
stackable!



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

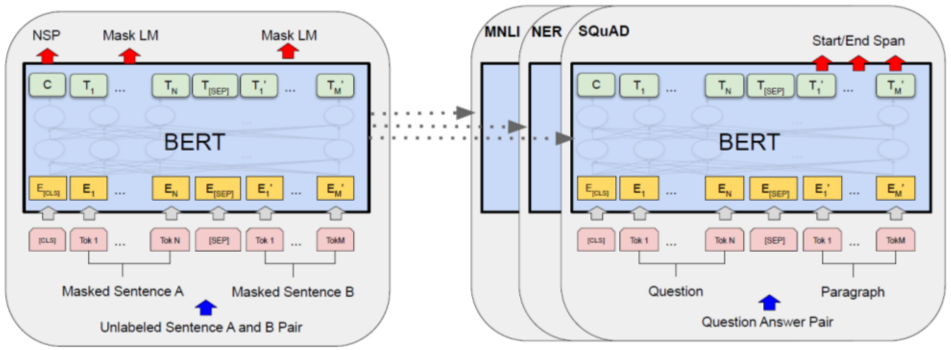
BERT (Bidirectional Encoder Representations from Transformers)

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



Pre-training
Fine-Tuning

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

Input: [CLS] my dog is cute [SEP] he likes play ##ing [SEP]

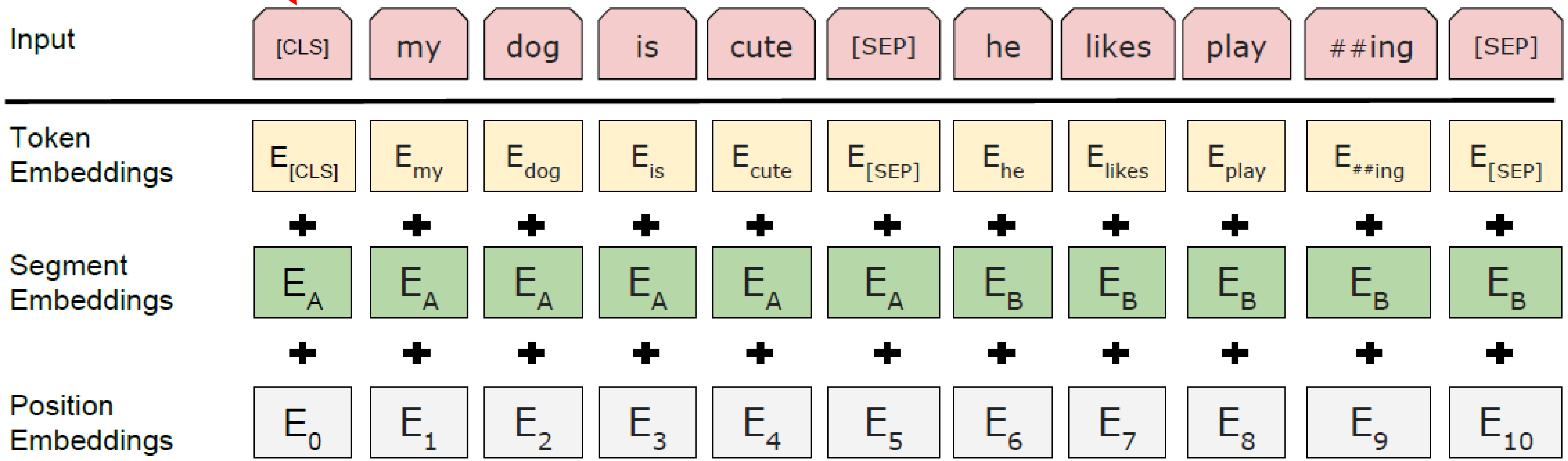
Devlin, etal, 2019

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)

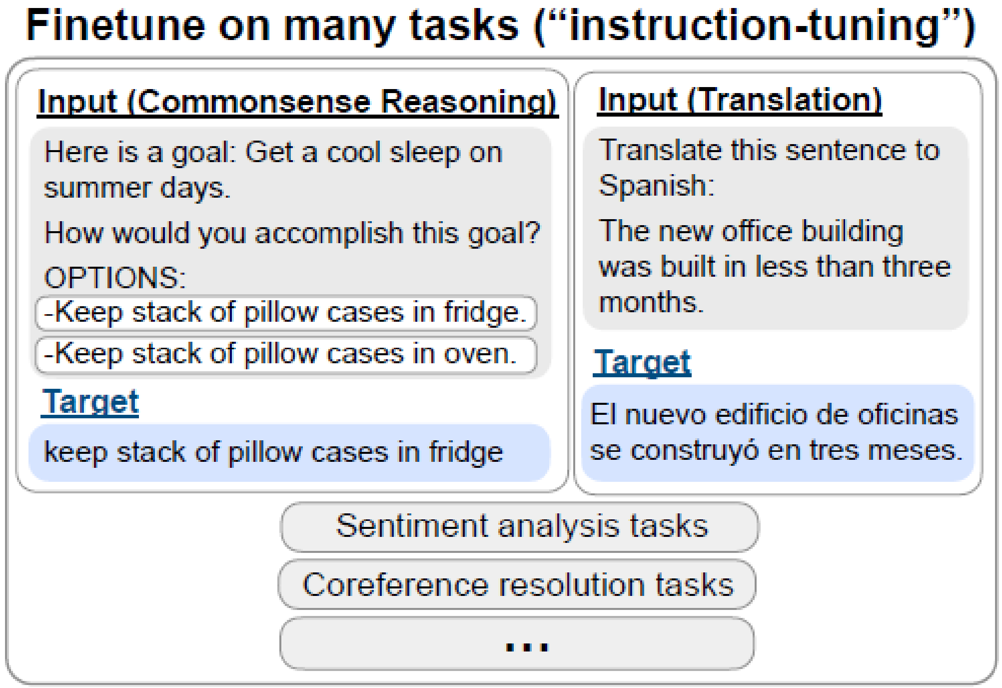
Goal: Train a transformer model at large scale so that it develops very general representations that are useful for many language tasks.

‘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,

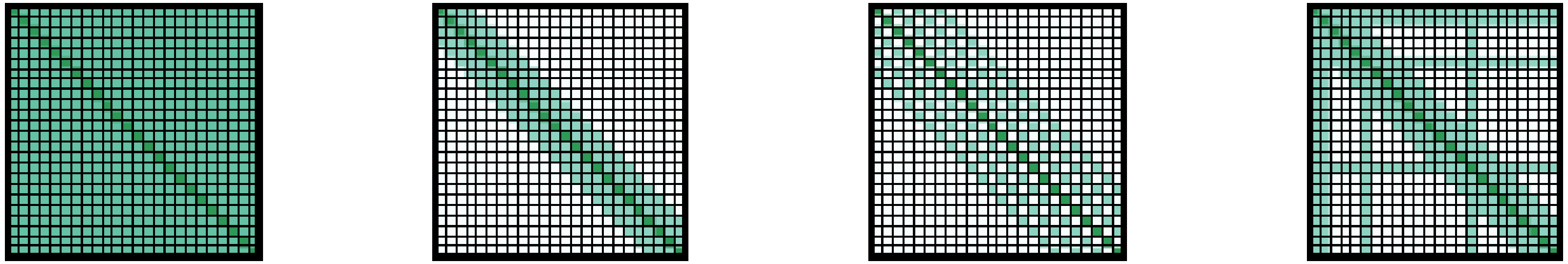
‘Instruction tuning’ improves models so they don’t need examples

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?

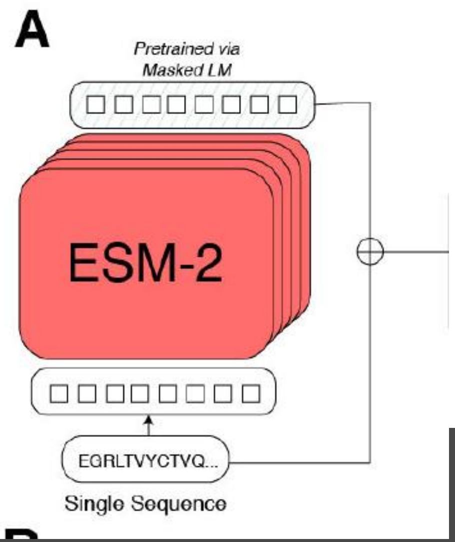
ESM Fold model

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)

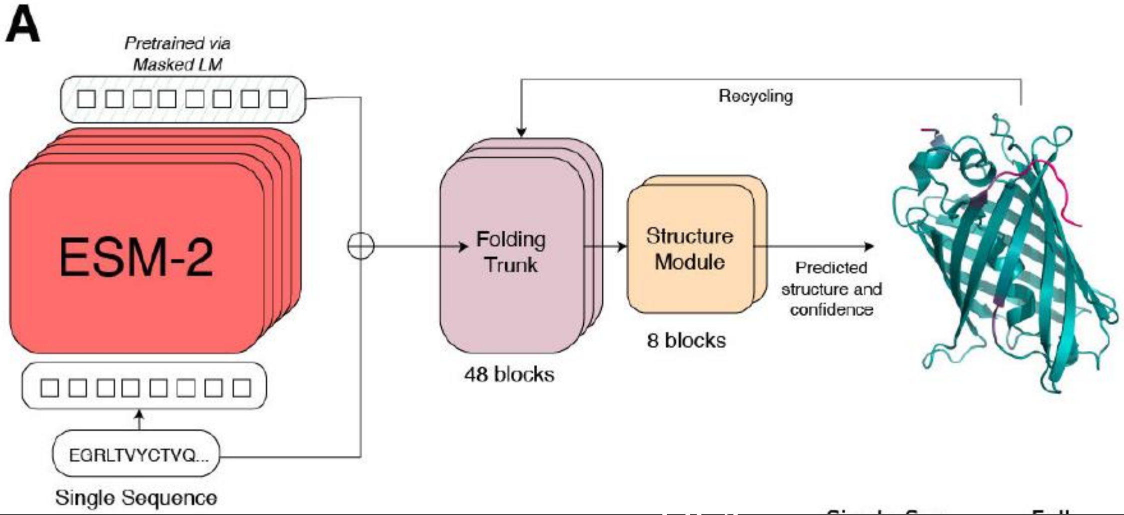


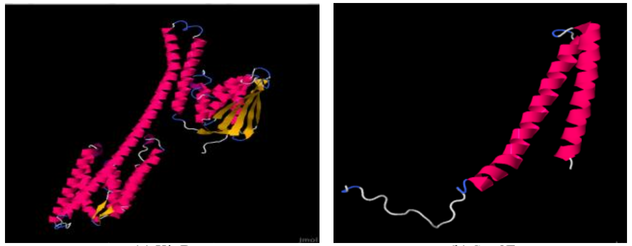
ESM Fold model

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)





(a) KinB (b) Spo0E

Figure 1. Protein Structures.

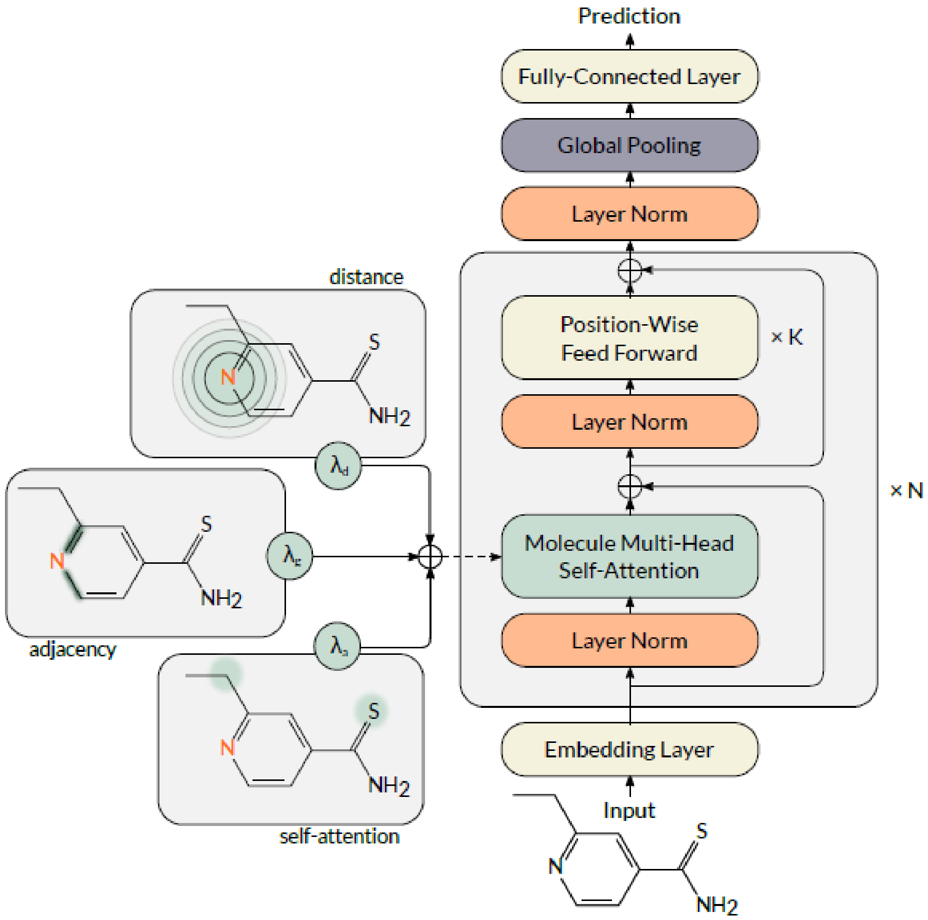
The two protein structures presented here KinB and Spo0E were generated by providing sequence of ~300 amino acids to ESMFold. (M.Gujral SDSC)

Molecule Attention Transformer
(Maziarka et al. 2020)

Molecular property prediction

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

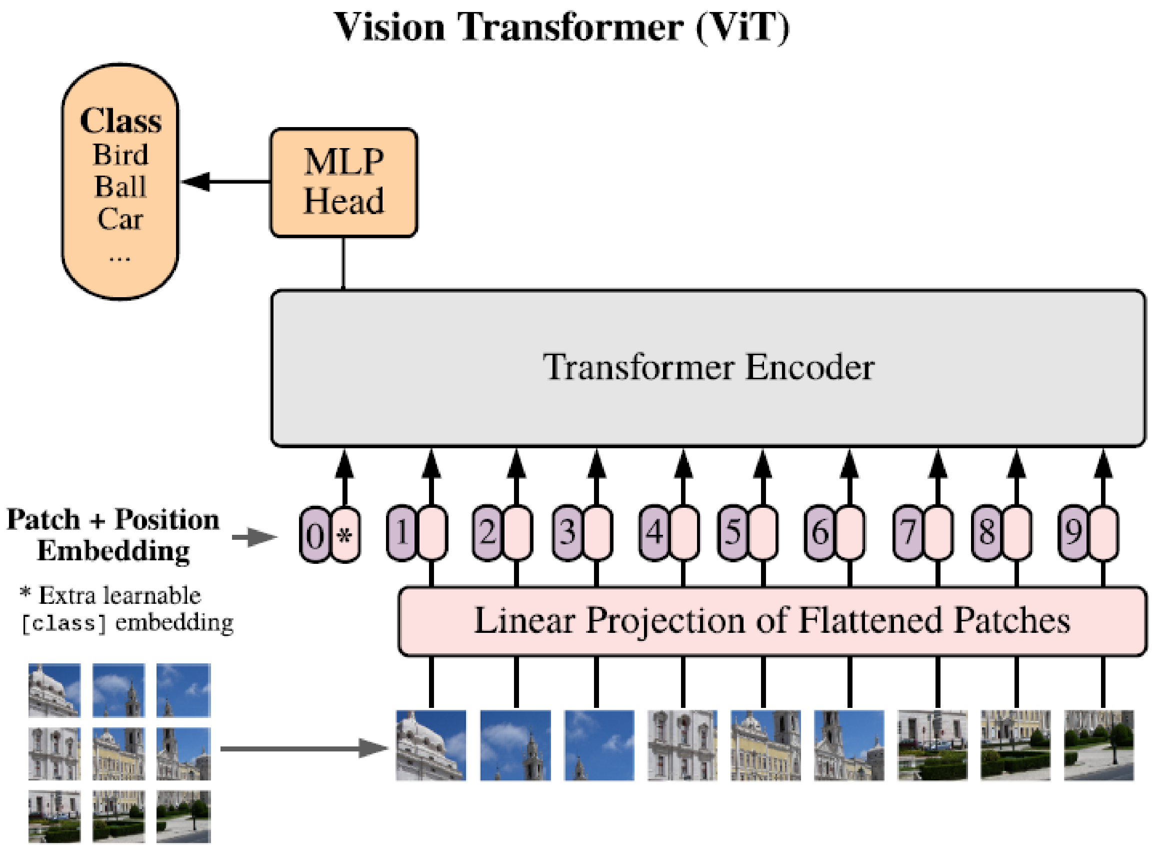
Includes spatial information by using a sum of the attention matrix, a distance matrix, and an adjacency matrix.



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)

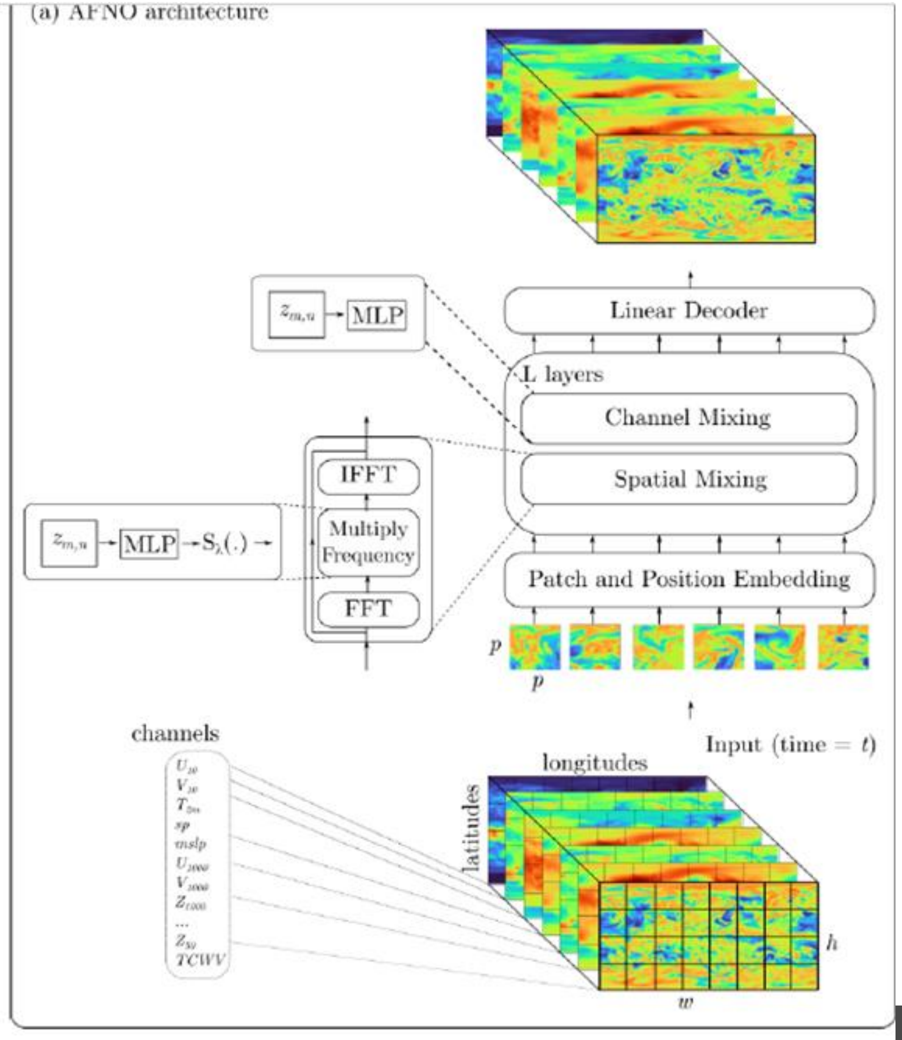
Uses a classification token like Bert to learn image output classes

Competitive or better than CNNs but might need more data

Adaptive Fourier Neural Operator for weather prediction

Luca Delle Monache, Pat Mulrooney, Agniv Sengupta, SIO, UCSD

- Like a ViT starts with image patch and position embedding of atmospheric images
- Then uses FFT for ‘embedding/encoding’
- Instead of ‘attention’ matrix is uses an MLP to model interdependencies (mixing) between FFT components
- (It’s not exactly a transformer but a similar strategy. Also FFT for physical processes makes sense.



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

CLIP – Contrastive Language-Image Pretraining

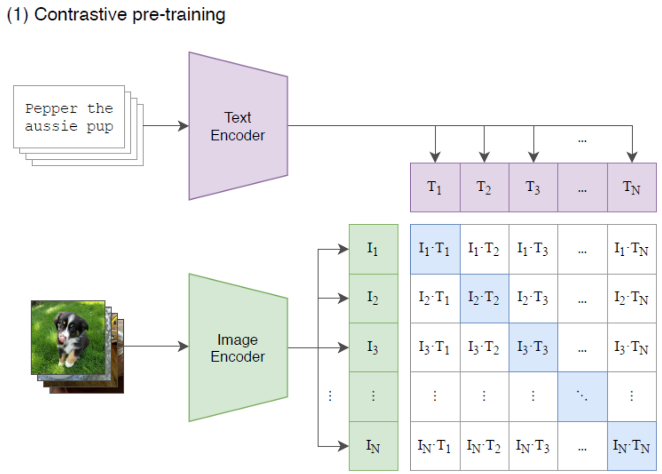
Learning Transferable Visual Models From Natural Language Supervision

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

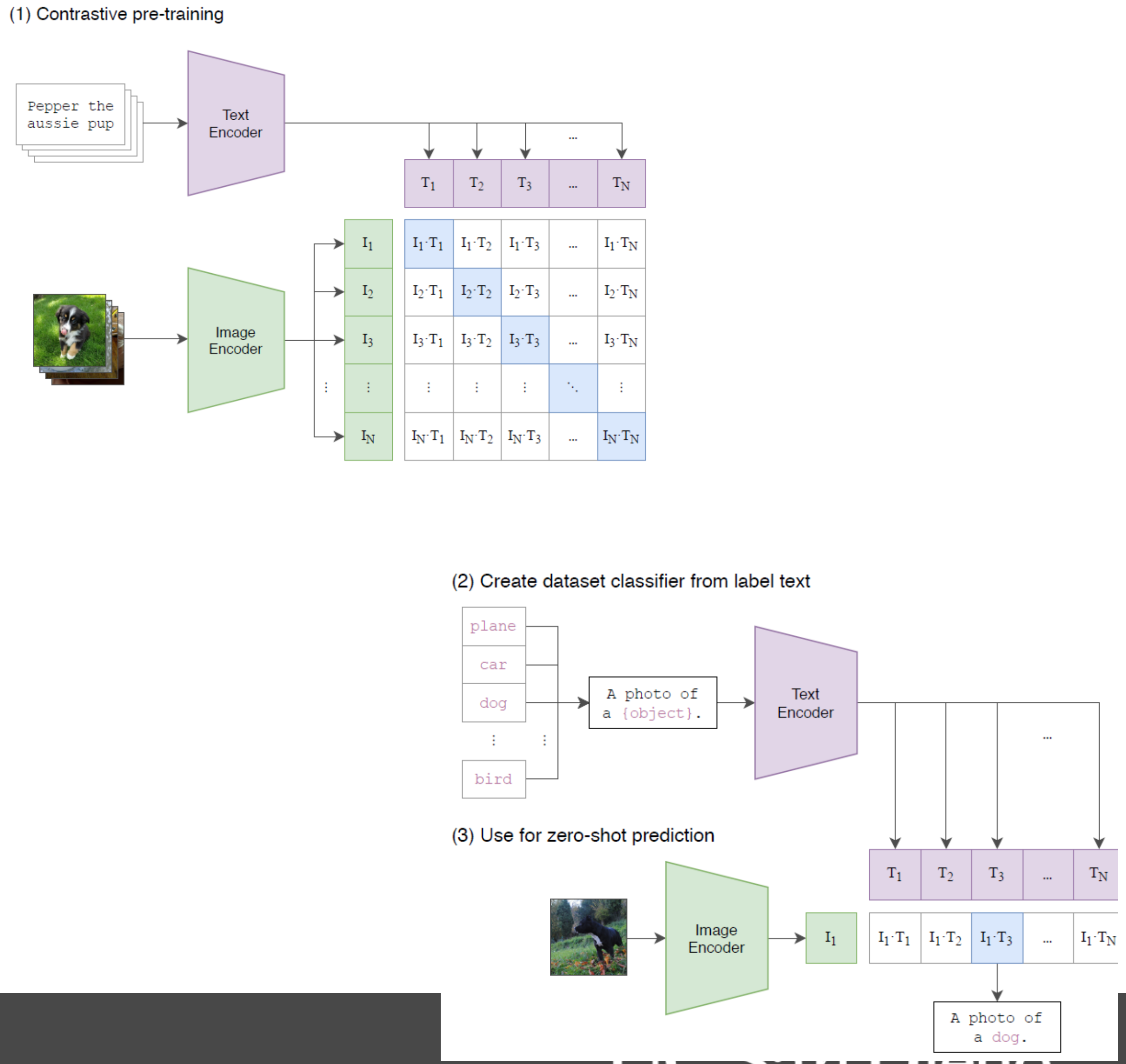
Uses 400M images and captions for training

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



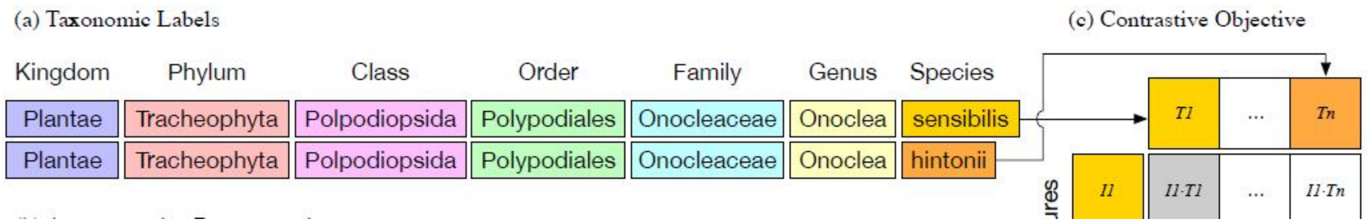
BIOCLIP: A Vision Foundation Model for the Tree of Life

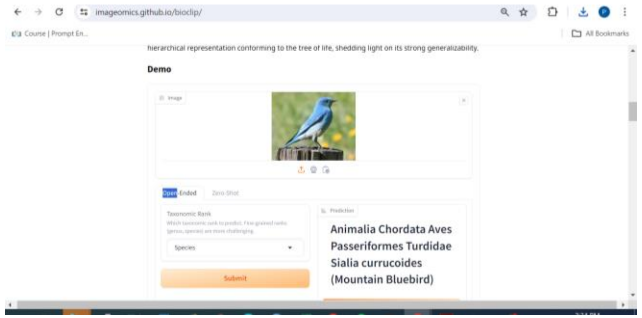
Stevens, etal 2024 OSU

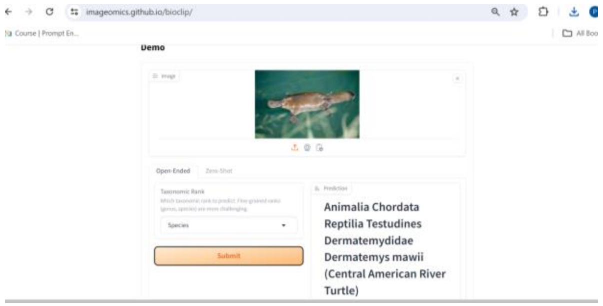
Uses pre-trained CLIP for a base

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 use jupyter-gpu-shared-pytorch
2. start singularity shell
3. pip install from github to a user folder
4. Export PYTHONPATH
5. 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