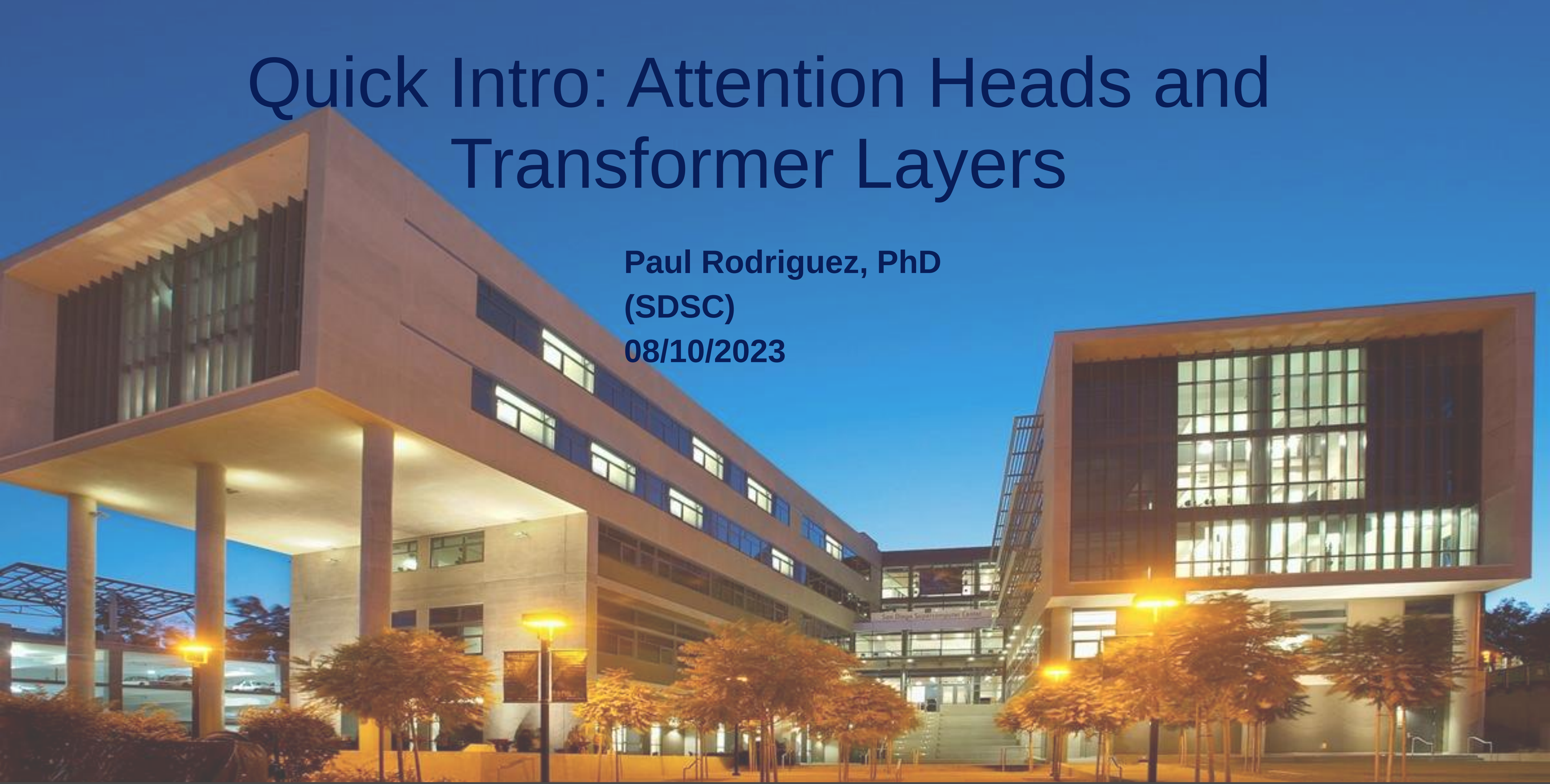
Quick Intro: Attention Heads and Transformer Layers

Paul Rodriguez, PhD
(SDSC)
08/10/2023



Outline

- Basic word prediction task and motivating the attention strategy
- A basic Attention Head network
- Transformers
- Exercise: Working with BERT pretrained transformer model

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’?

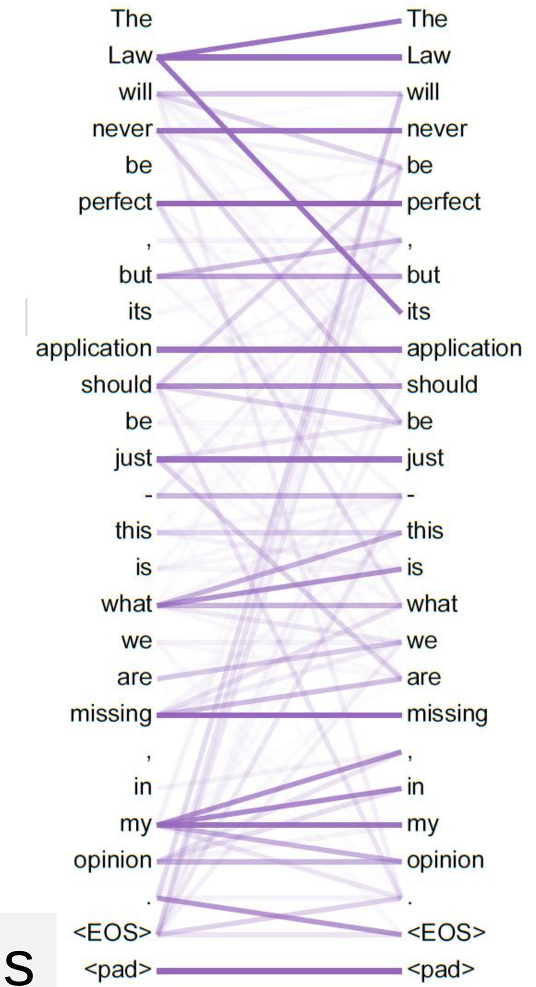
Dependences of Language

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 data sample:

  <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	Ordrd
0	<start>		1.0			
1	The			0.5	0.5	
2	Man					1.0
3	Ordrd		1.0			
4	The			0.5	0.5	
5	Chickn	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	Ordrd
0	<start>		1.0			
1	The			0.5	0.5	
2	Man					1.0
3	Ordrd		1.0			
4	The			0.5	0.5	
5	Chickn	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=TxV* has contextual predictive information

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

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

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=TxV* has contextual predictive information

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

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

W will be learned and should depend on transformations of X

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

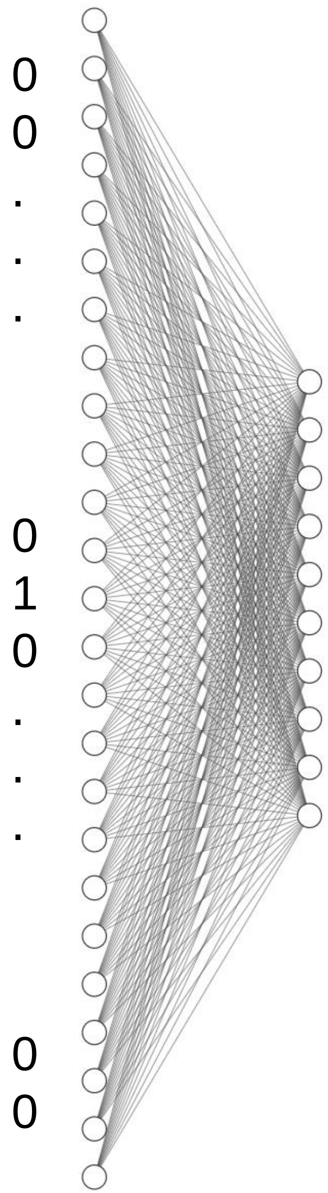
$$W_{dep} \odot Mask = \begin{pmatrix} w_{11} & 0 & \cdots & \cdots & 0 \\ w_{21} & w_{22} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{T1} & w_{T2} & w_{T3} & \cdots & w_{TT} \end{pmatrix}$$

Finally, apply Softmax to each row to make it like probability weights

Transformation as Embedding Layer/Table

Let X be given as a 'one-hot vector input' – where ***only one*** of the 25 input nodes is 1, the rest are 0.

X is 25x1

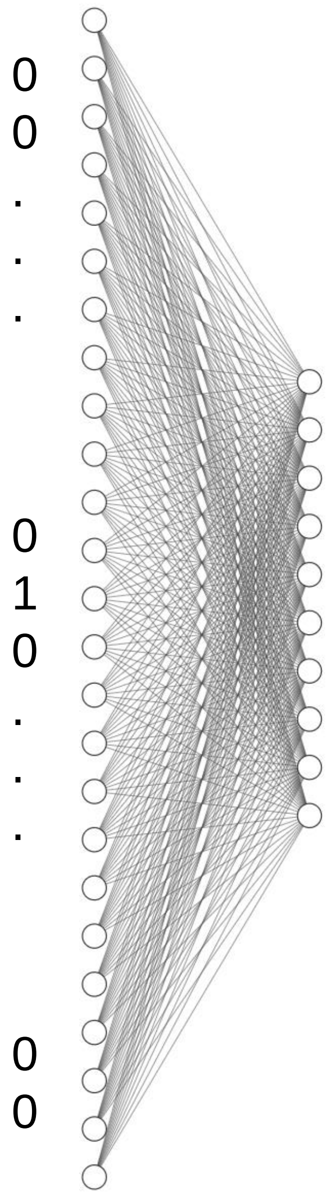


Transformation as Embedding Layer/Table

Let *X* be given as a 'one-hot vector input' – where ***only one*** of the 25 input nodes is 1, the rest are 0.

Let W be 10x25 matrix, and let $Y = W * X$

X is 25x1



Y is 10x1

Transformation as Embedding Layer/Table

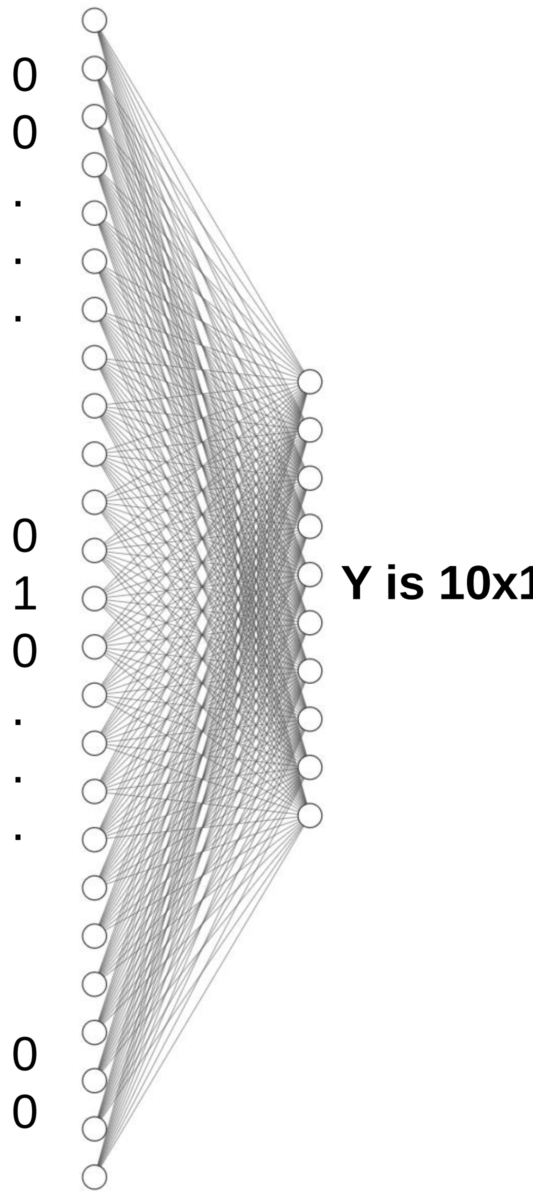
Let X be given as a ‘one-hot vector input’ – where ***only one*** of the 25 input nodes is 1, the rest are 0.

Let W be 10x25 matrix, and let $Y = W * X$

if $X_j = 1$ then: $Y = W * X$

$$= \begin{pmatrix} | & | & . & | & . & . \\ w1 & w2 & . & w_j & . & . \\ | & | & . & | & . & . \end{pmatrix} * \begin{pmatrix} 0 \\ 0 \\ . \\ 1 \\ . \\ . \end{pmatrix} = ?$$

X is 25x1



Transformation as Embedding Layer/Table

Let X be given as a 'one-hot vector input' – where ***only one*** of the 25 input nodes is 1, the rest are 0.

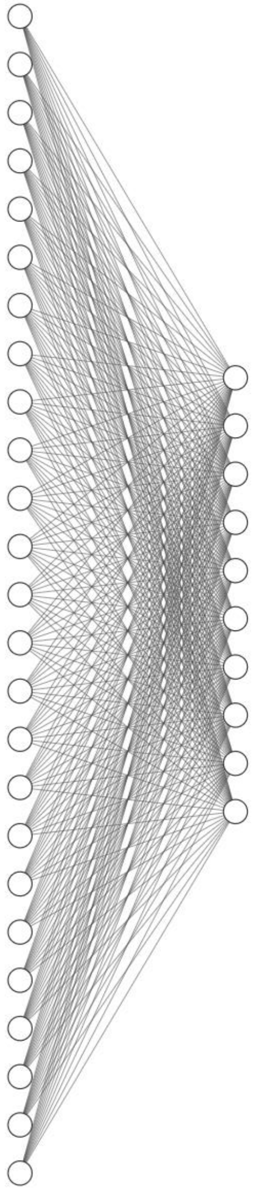
Let W be 10x25 matrix, and let $Y = W * X$

if $X_j = 1$ then: $Y = W * X$

$$= \begin{pmatrix} | & | & . & | & . & . \\ w1 & w2 & . & w_j & . & . \\ | & | & . & | & . & . \end{pmatrix} * \begin{pmatrix} 0 \\ 0 \\ . \\ 1 \\ . \\ . \end{pmatrix} = \begin{pmatrix} | \\ w_j \\ | \end{pmatrix}$$

X is 25x1

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



Y is 10x1

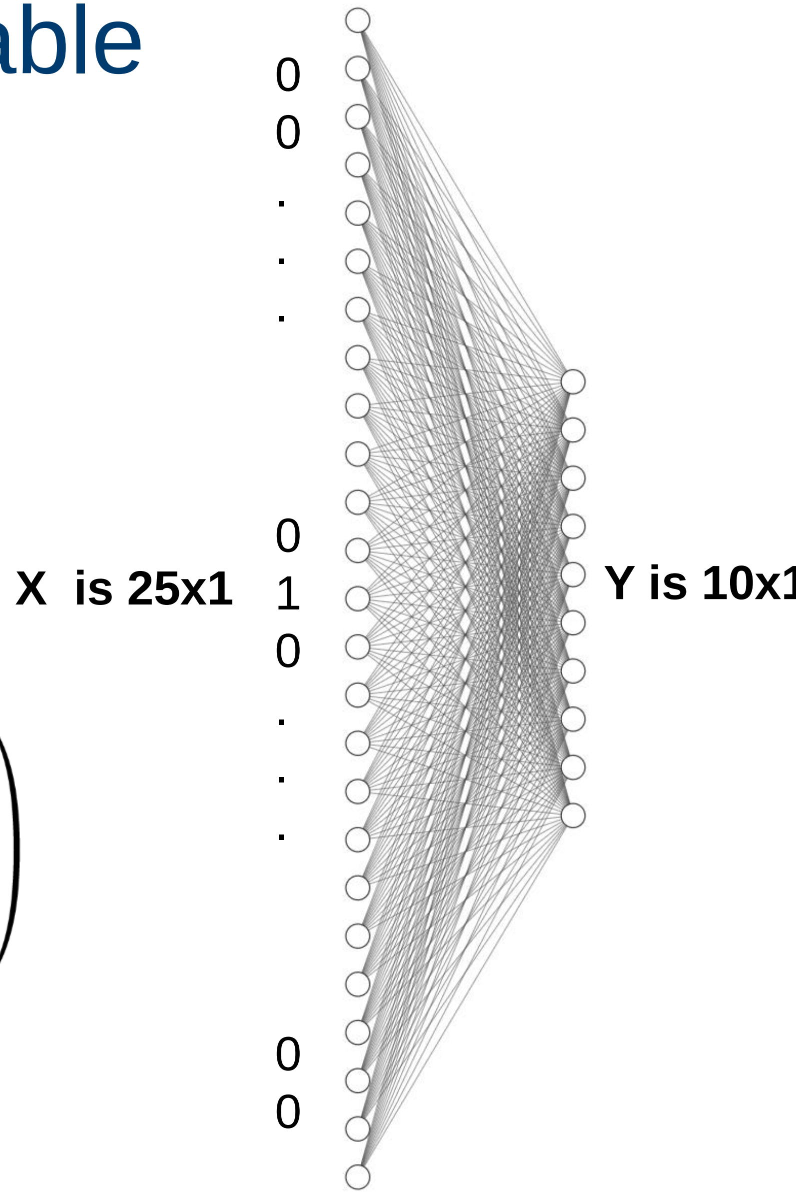
Transformation as Embedding Layer/Table

Let X be given as a 'one-hot vector input' – where ***only one*** of the 25 input nodes is 1, the rest are 0.

Let W be 10x25 matrix, and let $Y = W * X$

if $X_j = 1$ then: $Y = W * X$

$$= \begin{pmatrix} | & | & \cdot & | & \cdot & \cdot \\ w1 & w2 & \cdot & w_j & \cdot & \cdot \\ | & | & \cdot & | & \cdot & \cdot \end{pmatrix} * \begin{pmatrix} 0 \\ 0 \\ \cdot \\ 1 \\ \cdot \\ \cdot \end{pmatrix} = \begin{pmatrix} | \\ w_j \\ | \end{pmatrix}$$



So just let X be sequence of token ids (1 to 25) and treat W like a table

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: $\underset{T\times H}{\mathrm{Q}} * \underset{H\times T}{\mathrm{K}'} \rightarrow \underset{T\times T}{\mathrm{W_{dep}}}$

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

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 a scaling by H

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

An example of attention head with a toy task:
(in the SI2023_toytask_attention notebook)

CIML23_toytask_attentionhea

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 that *the*→ *[chicken or beef]* predictions change b/c of context

```
1/1 [==============================] - 0s 106ms/step

 This is the output predictions use Head size 20
         <ST>    the    man   chkn  ordrd  woman   beef
0 <ST>  0.596  0.037  0.988  0.202  0.193  0.988  0.077
1 man   0.062  0.579  0.575  0.243  0.996  0.184  0.802
2 ordrd 0.092  0.991  0.122  0.612  0.319  0.050  0.403
3 the   0.225  0.705  0.016  0.996  0.008  0.018  0.837
4 chkn  0.994  0.103  0.365  0.679  0.104  0.634  0.353

 This is the output predictions use Head size 20
         <ST>    the    man   chkn  ordrd  woman   beef
0 <ST>  0.596  0.037  0.988  0.202  0.193  0.988  0.077
1 woman 0.037  0.578  0.189  0.061  0.999  0.051  0.940
2 ordrd 0.129  0.994  0.049  0.626  0.153  0.034  0.511
3 the   0.028  0.686  0.163  0.727  0.786  0.034  0.992
4 beef  0.994  0.107  0.369  0.679  0.099  0.633  0.340
```

Output TxV predictions:

Notice that *the* → *[chicken or beef]* predictions change b/c of context

```
1/1 [==============================] - 0s 106ms/step

 This is the output predictions use Head size 20
          <ST>    the    man   chkn  ordrd  woman   beef
0 <ST>   0.596  0.037  0.988  0.202  0.193  0.988  0.077
1 man    0.062  0.579  0.575  0.243  0.996  0.184  0.802
2 ordrd  0.092  0.991  0.122  0.612  0.319  0.050  0.403
3 the    0.225  0.705  0.016  0.996  0.008  0.018  0.837
4 chkn   0.994  0.103  0.365  0.679  0.104  0.634  0.333

 This is the output predictions use Head size 20
          <ST>    the    man   chkn  ordrd  woman   beef
0 <ST>   0.596  0.037  0.988  0.202  0.193  0.988  0.077
1 woman  0.037  0.578  0.189  0.061  0.999  0.051  0.940
2 ordrd  0.129  0.994  0.049  0.626  0.153  0.034  0.511
3 the    0.028  0.686  0.163  0.727  0.786  0.034  0.992
4 beef   0.994  0.107  0.369  0.679  0.099  0.633  0.340
```

TxT Attn Wts:

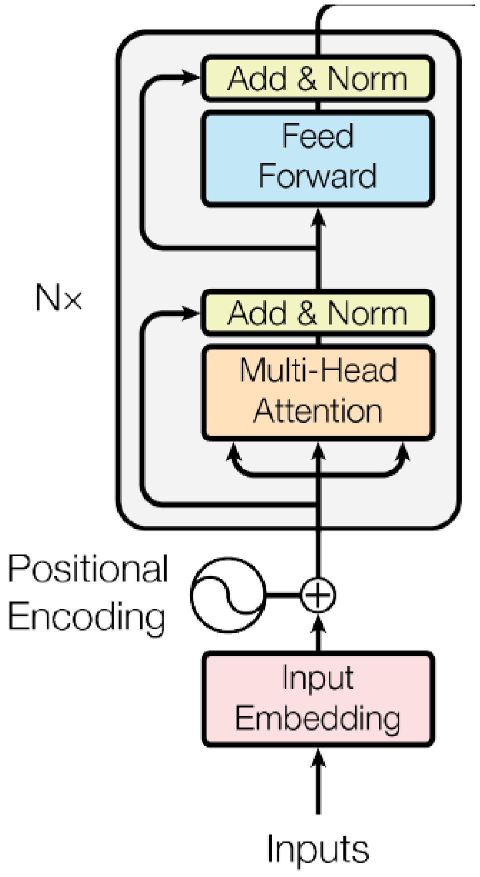
Notice that 'the' predictions depend on 'ordered' ,'man', and/or 'woman'

```
for i-th input: 0   These are the output at layer  AttnWts
         0 <ST>  1 man   2 ordrd  3 the   4 chkn
0 <ST>    1.000  0.000    0.000  0.000   0.000
1 man     0.093  0.907    0.000  0.000   0.000
2 ordrd   0.064  0.561    0.375  0.000   0.000
3 the     0.003  0.213    0.759  0.025   0.000
4 chkn    0.258  0.003    0.160  0.523   0.056
The head size H was:  20
for i-th input: 1   These are the output at layer  AttnWts
         0 <ST>  1 woman  2 ordrd  3 the   4 beef
0 <ST>    1.000    0.000    0.000  0.000   0.000
1 woman   0.079    0.921    0.000  0.000   0.000
2 ordrd   0.091    0.374    0.535  0.000   0.000
3 the     0.001    0.769    0.223  0.007   0.000
4 beef    0.252    0.001    0.162  0.526   0.059
The head size H was:  20
```

Finally, a transformer

Include skip-add connections

Include Layer Normalization or DropOut layers

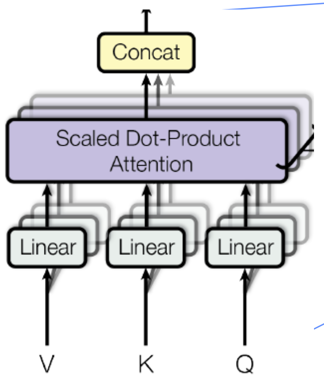


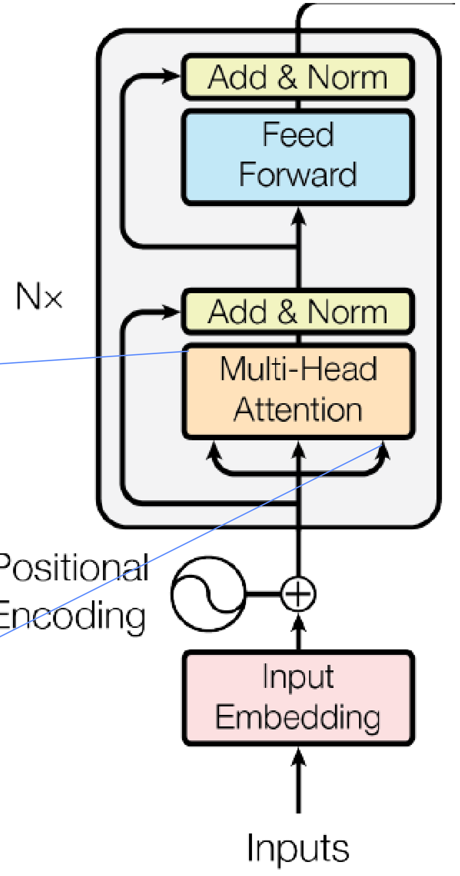
Finally, a transformer

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

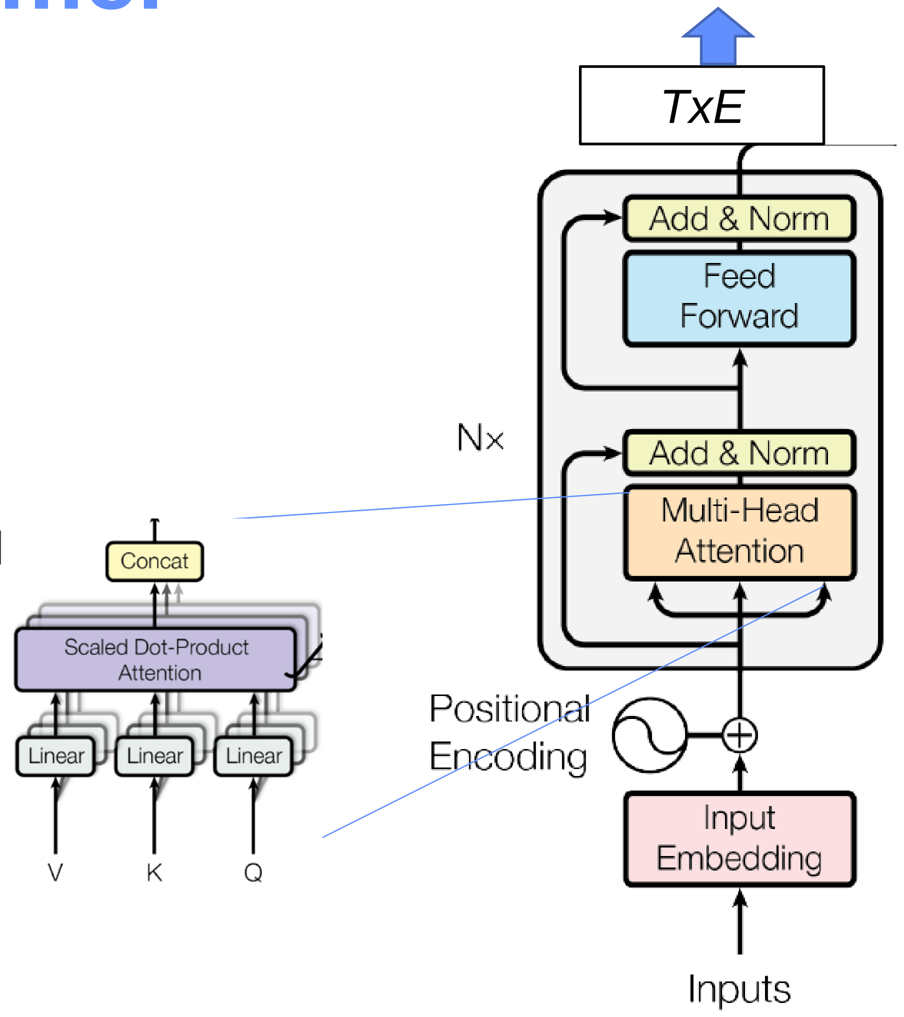
Include Layer Normalization or DropOut layers

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

Add MLP layers on top –

output another *TxE* matrix

stackable!

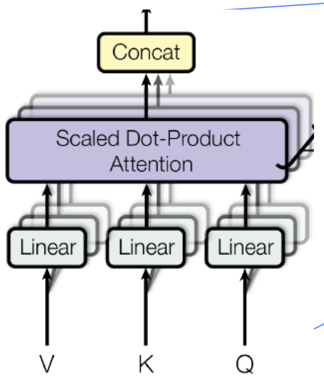


Finally, a transformer

Include skip-add connections

Include Layer Normalization or DropOut layers

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

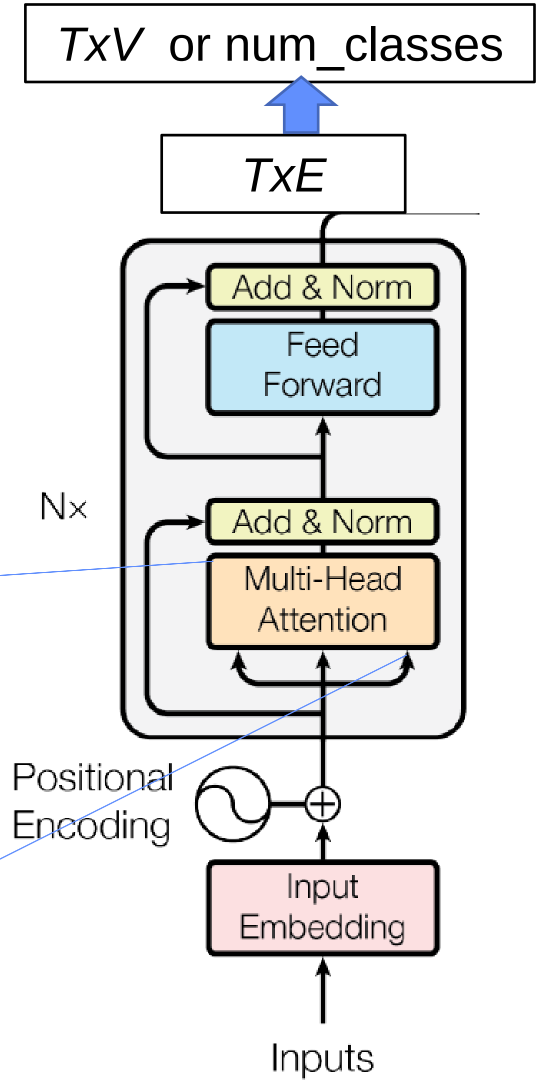


Add MLP layers on top –

output another *TxE* matrix

or output final probabilities

stackable!



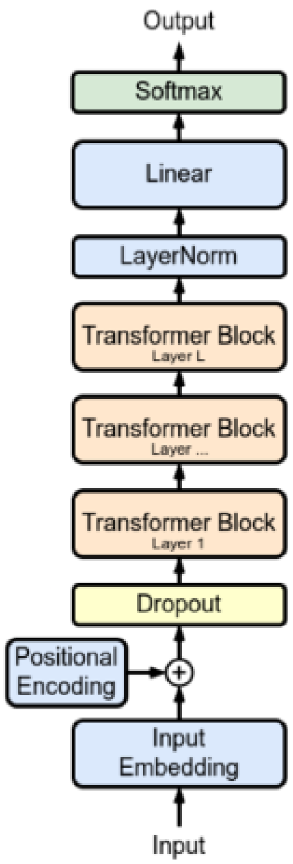
2 kinds of training strategies

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

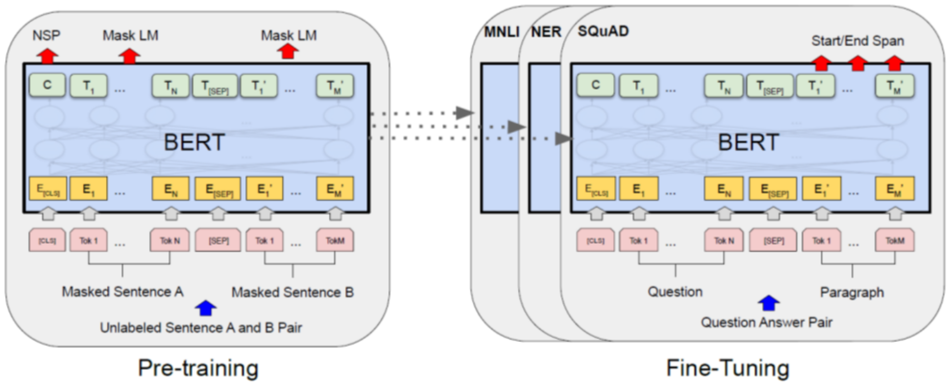


BERT (Bidirectional Encoder Representations from Transformers)

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

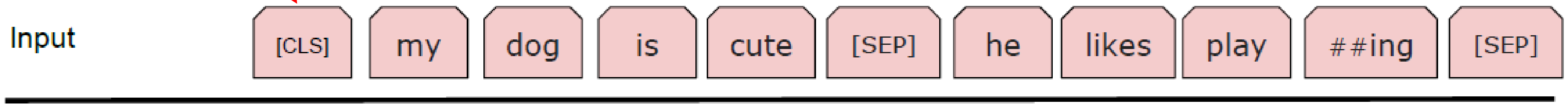
2 Fine tune on variety of tasks

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



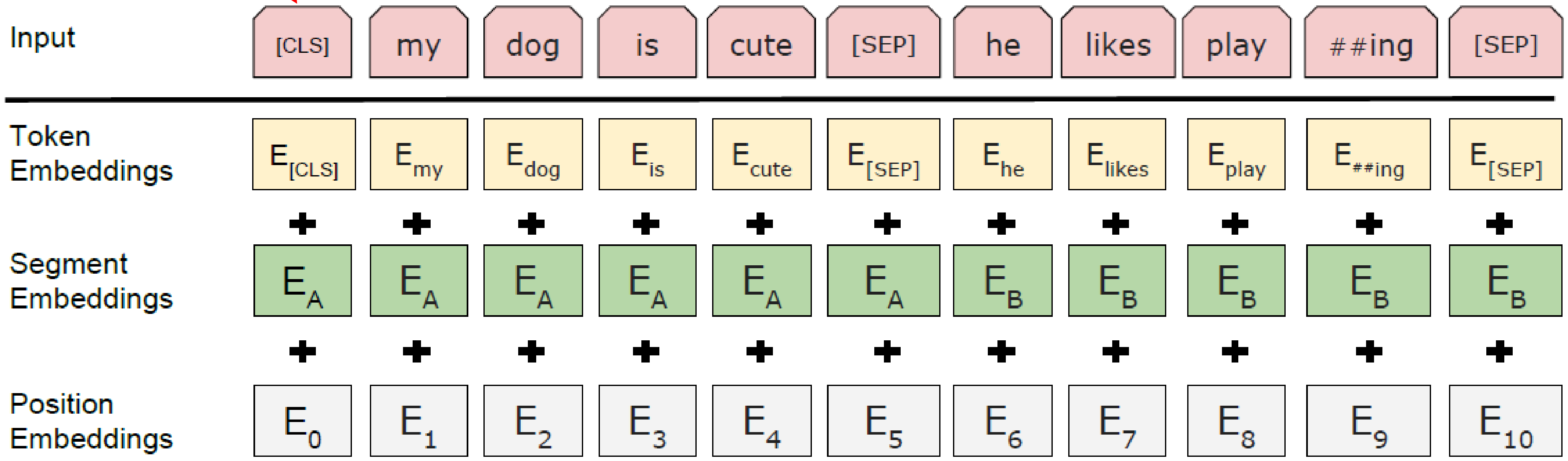
Devlin, etal, 2019

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

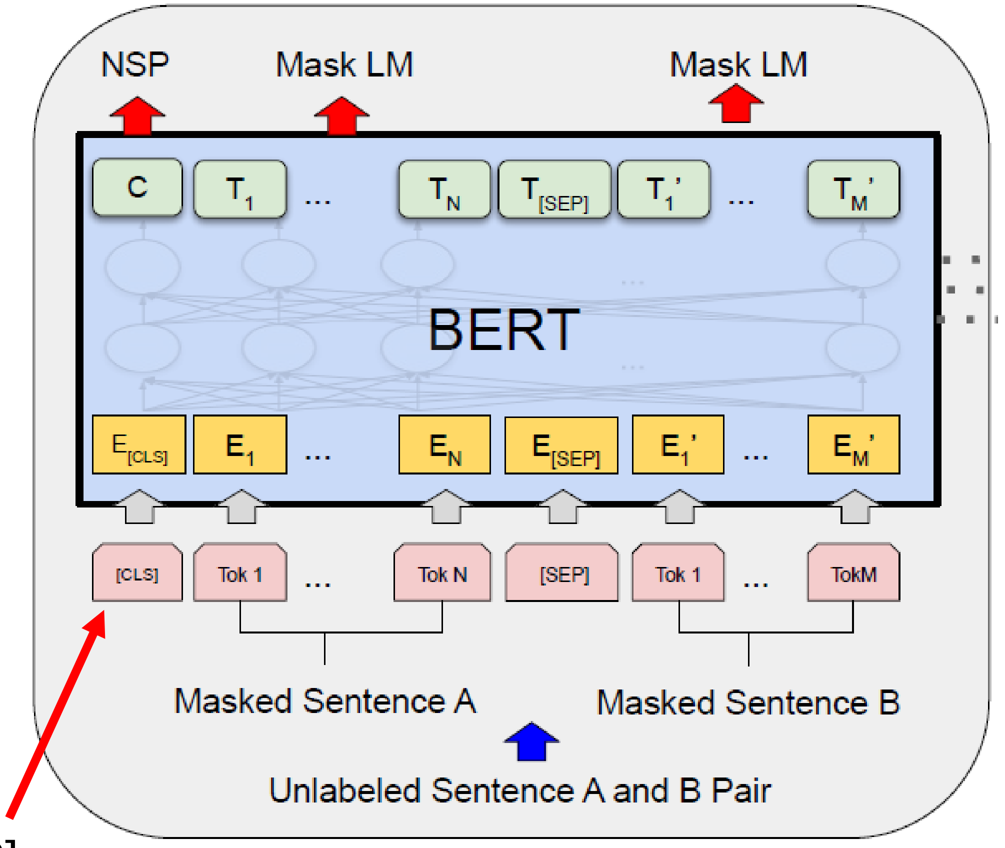
BERT –

Last layer *TxV* output

Transformer Layers

Embeddings

T Input tokens



[CLS]
is'dummy' token
and always
present

Pre-training

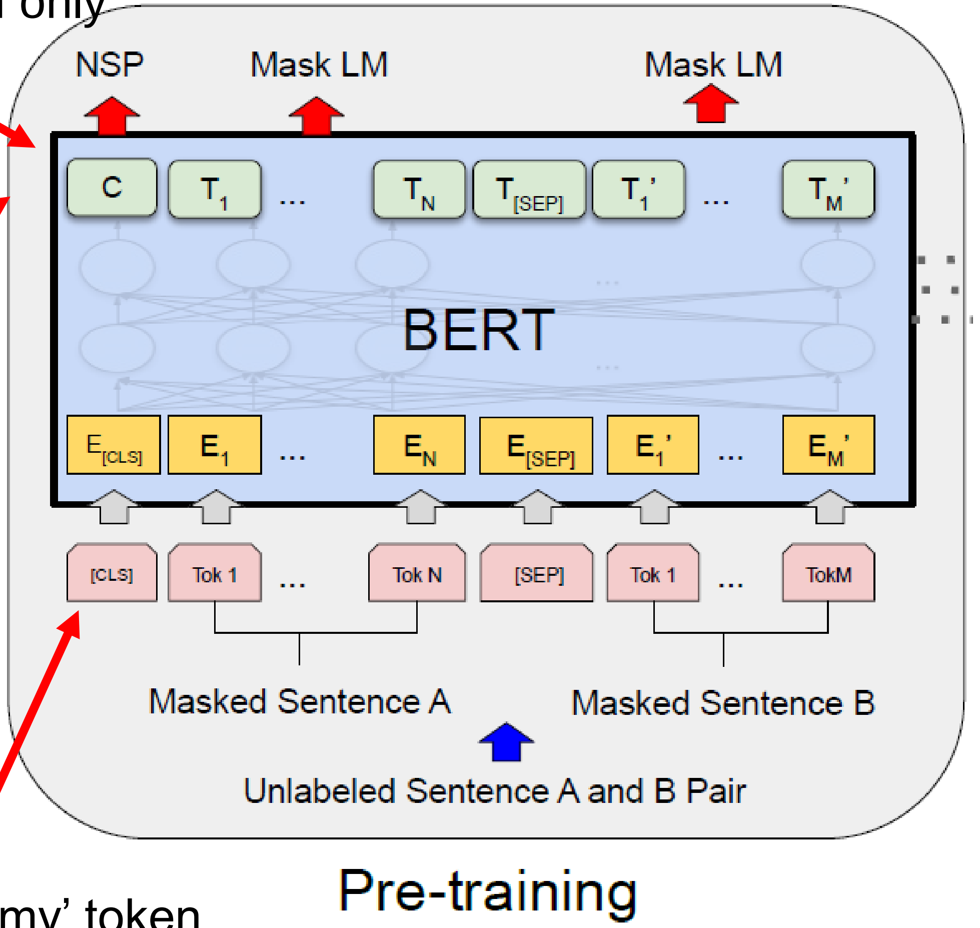
Devlin, etal, 2019

Train a classification task on this [CLS] token prediction only

BERT –

For classification over paired sentences or any text, train C row vector (in the last layer *TxV* output matrix)

"NSP": classify if next sentence follows first sentence



[CLS] is'dummy' token and always present

Pre-training

KerasNLP

- **KerasNLP is an extension to Keras**

- **KerasNLP has several pre-trained LLMs (large language models). Each model comes with related modules, for example:**

  - GPT2Backbone   the model without task specific output layers
  - GPT2CausalLM   the model with output predictions
  - GPT2CausalLMPreprocessor the preprocessor that feeds model.fit

KerasNLP

- **KerasNLP is an extension to Keras**

- **KerasNLP has several pre-trained LLMs (large language models). Each model comes with related modules, for example:**

  - GPT2Backbone the model without task specific output layers
  - GPT2CausalLM the model with output predictions
  - GPT2CausalLMPreprocessor the preprocessor that feeds model.fit

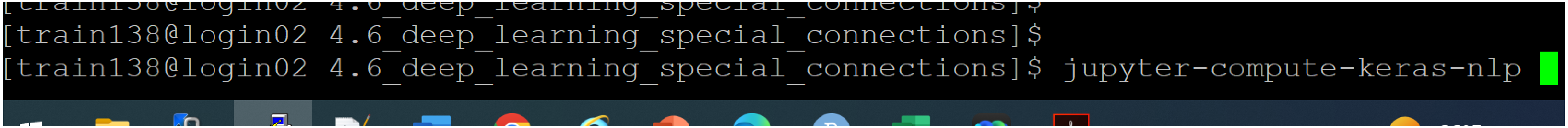
We will use pre-trained BERT and compare different BERT versions

Notebook exercise using KerasNLP

- In a terminal window start the notebook session for keras-nlp

  …]$ cd 5.3b-deep_learning_part2/special_connections

  …]$ jupyter-compute-keras-nlp



- Open the URL and look for the *SI2023_BERT_FineTune_v3.ipynb* notebook

Keras NLP package has several BERT versions

So let's start with 'bert-small' (28M parameters)

For reference:
BERT BASE (L=12, H=768, Attn=12, Total Parameters=110M)
BERT LARGE (L=24, H=1024, Attn=16, Total Parameters=340M).

models/

bert_tiny_en_uncased	BERT	4M	2-layer BERT model where all input is lowercased. Trained on English Wikipedia + BooksCorpus.
bert_small_en_uncased	BERT	28M	4-layer BERT model where all input is lowercased. Trained on English Wikipedia + BooksCorpus.
bert_medium_en_uncased	BERT	41M	8-layer BERT model where all input is lowercased. Trained on English Wikipedia + BooksCorpus.
bert_base_en_uncased	BERT	109M	12-layer BERT model where all input is lowercased. Trained on English Wikipedia + BooksCorpus.
bert_base_en	BERT	108M	12-layer BERT model where case is maintained. Trained on English Wikipedia + BooksCorpus.
bert_base_zh	BERT	102M	12-layer BERT model. Trained on Chinese Wikipedia.
bert_base_multi	BERT	177M	12-layer BERT model where case is maintained. Trained on trained on Wikipedias of 104 languages

END