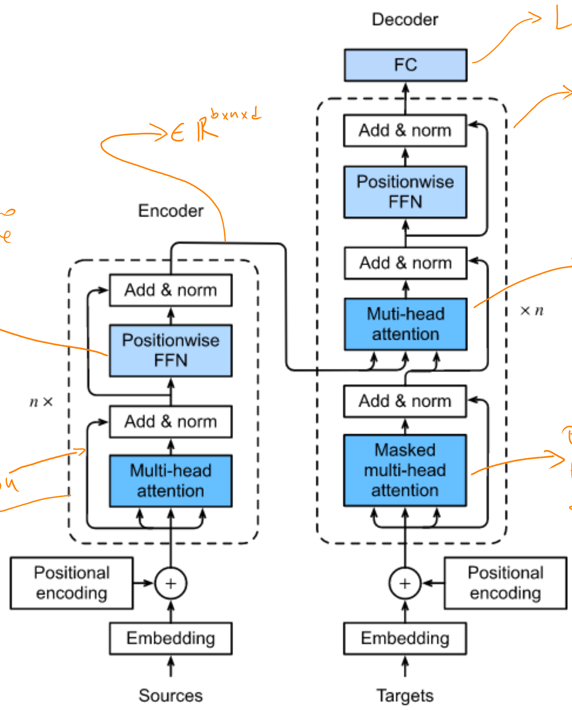
TRANSFORMER REVIEW



→ Linear layer and softmax

→ Runs multiple times to produce a predicted word at each step.

→ $\in \mathbb{R}^{b \times n \times d}$

It seems this learns complex interactions of words:
- while wire
- keep up
- get angry

Applies transformations to the representation of the sequences.
It is an MLP

Encoder-decoder attention

Inputs:
- x from prev. decoder layer (query)
- m (memory) from the output of the encoder. Twice as key and value.

Residual connection

The encoder run once per batch

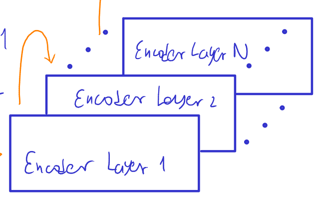
Queries, keys, values from outputs of prev. decoder layer except the first layer which is the target

$X \in \mathbb{R}^{b \times n \times d}$ $Y \in \mathbb{R}^{b \times n \times d}$

LEGEND:

b: Batch size
n: sequence length
d: Embedding dimensionality
h: Number of heads

THE ENCODER



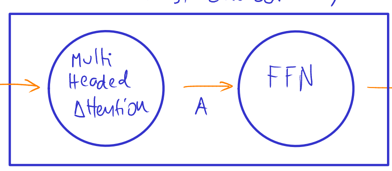
Encoder Layer 1 output as input for Layer 2

Context and meaning already embedded on X.

$X_{b,n,d}$ →

→ Encoder output Shape (b,n,d) to Decoder for the Encoder-decoder Attention as "memory" as key and value.

First Encoder Layer

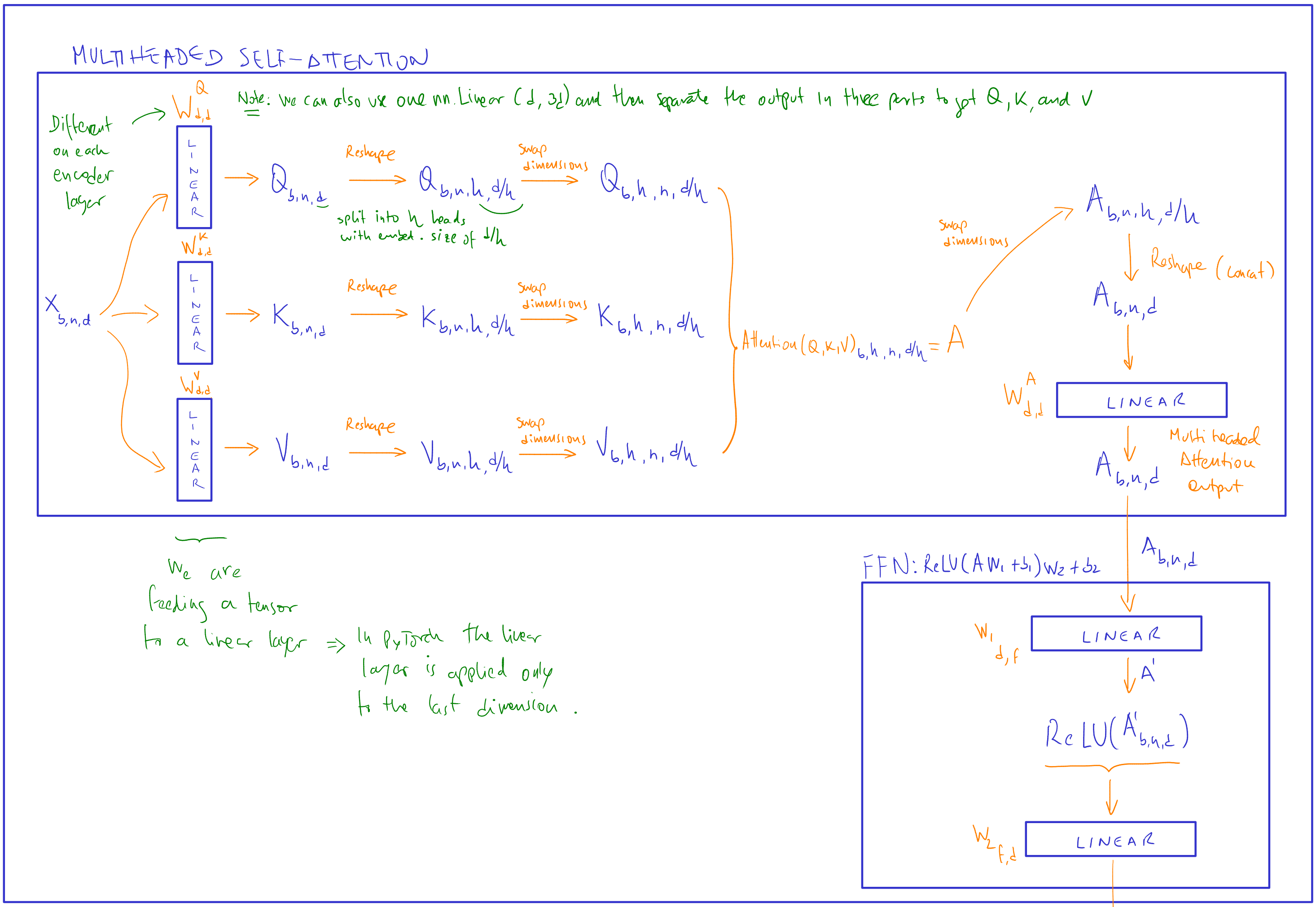


$X_{b,n,d}$ →

→ Encoder output to next encoder layer

A CLOSER LOOK TO THE ENCODER

This is the first encoder layer

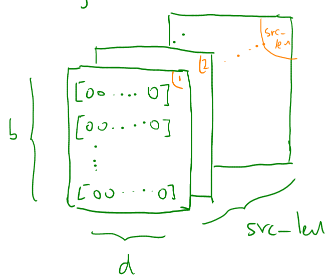


This will replace X as input in next encoder layer < Encoder output to next encoder layer : (b,n,d)

A CLOSER LOOK TO THE ATTENTION MECHANISM

THE INPUTS TO ATTENTION:

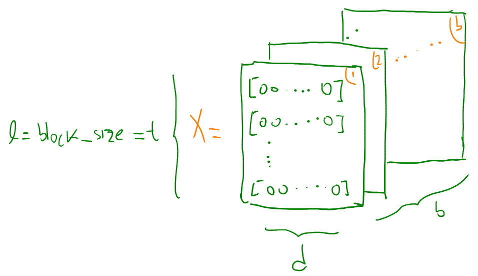
In RNNs we were interested in processing batches of the i-th word of each sentence at each timestep something like this:



where:

- $b \to$ A batch of all i-th words of all sentences
- $d \to$ The embedding size for each word
- src_len $\to$ The len of the longest sentence in the input dataset.

With transformers we can pass a batch of entire sentences, so the inputs might look like this:



where:

- b: The # of sentences in the input dataset
- l: The # of words of sentence i of the dataset
- d: The dimension of each word embedding in the sentence

This is a tensor of shape (b, l, d) (rows, each word) when applying softmax over this tensor we do it row-wise, meaning, the second dimension

When transposing this tensors we need to forget the batches for a moment and just think about the dimensions independently of the batch dimension. It is not a good idea using the T operand in PyTorch since

$(b, \ell, d)^T \to (d, \ell, b)$. Better $(\underbrace{b, \ell, d}_{\text{switch}}).\text{transpose}(1,2) \Rightarrow (b, d, \ell)$ so we can do @ with torch.bmm:

$(\underbrace{b, \ell, d) \times (b, d}_{\text{bridge}}, \ell) \to (b, \ell, \ell)$

As we are working with self attention each input vector x_i (a single token?) assumes 3 roles:

- Keys
- Queries
- Values

To make things easier we'll define 3 new vectors as linear transformations of x_i through 3 $d \times d$ matrices: W_k, W_q, W_v

$q_i = W_q x_i \qquad k_i = W_k x_i \qquad v_i = W_v x_i$

Shapes:

$\underset{(d,1)}{k_i} = \underset{(d,d)}{W_k}\, \underset{(d,1)}{x_i}$

Attention Layer

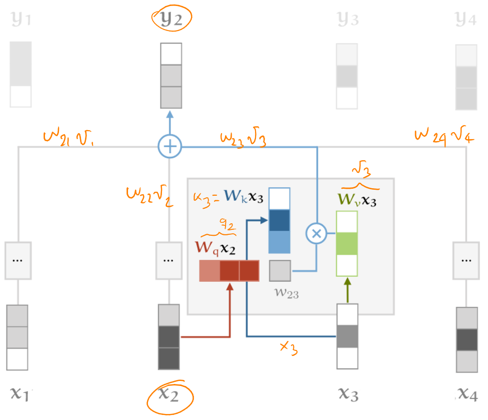
$$\boxed{\begin{aligned} &\text{Attention output: } y_i = \sum_j w_{ij} v_j \\ &\text{Attention distribution: } w_{ij} = \text{softmax}(w'_{ij}) \\ &\text{Attention scores: } w'_{ij} = q_i^T k_j \end{aligned}}$$

$\to y_i$ of shape $(e \times 1)$

x_i as $W_q x_i = q_i$ of shape $(d \times 1)$

To avoid gradient explosion we clip the attention scores

$$w'_{ij} = \frac{q_i^T k_j}{\sqrt{e}}$$



$w'_{23} = q_2^T k_3 = W_q x_2^T W_k x_3$

$w_{23} = \text{softmax}(W_q x_2^T W_k x_3)$

$y_2 = \underset{\text{scalar}}{w_{21}} v_1 + w_{22} \underset{\text{vector}}{v_2} + w_{23} v_3 + w_{24} v_4$

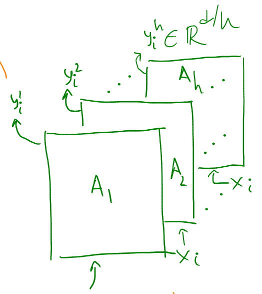
As human communication is complex we can't not capture the entire meaning with one attention pass. We need to stack more layers! $\Rightarrow$ Multi-headed Attention.

MULTI-HEADED SELF ATTENTION

We just "stack" several Self-Attention mechanisms: we'll have a set of weight matrices per layer

We actually split the d-dim. inputs in d/h heads

$\Rightarrow \underbrace{W_q^h \quad W_k^h \quad W_v^h}_{\text{Attention Heads}}$ where h is the # of layers.



$\Rightarrow \underbrace{[y_i^1 \; y_i^2 \cdots y_i^h]}_{\text{concatenation}} \in \mathbb{R}^d$

$\hookrightarrow \underset{W_{d,d}}{\boxed{\text{LINEAR TRANSF.}}} \to y_i \in \mathbb{R}^d$

$x_i \in \mathbb{R}^d$

$X_{\ell \times d} = \begin{bmatrix} x_1 \\ \vdots \\ x_\ell \end{bmatrix}$

This is expensive $\Rightarrow$ It is better to:

Not sure about this representation

$W_q^1 \; W_q^2 \cdots W_q^h \Rightarrow W_q \in \mathbb{R}^{d \times d}$

$W_k^1 \; W_k^2 \cdots W_k^h \Rightarrow W_k \in \mathbb{R}^{d \times d}$

$\underbrace{W_v^1 \; W_v^2 \cdots W_v^h}_{\text{concatenation}} \Rightarrow W_v \in \mathbb{R}^{d \times d}$

$\downarrow$ just 3 matrices

Better way to handle weights for multi headed Attention

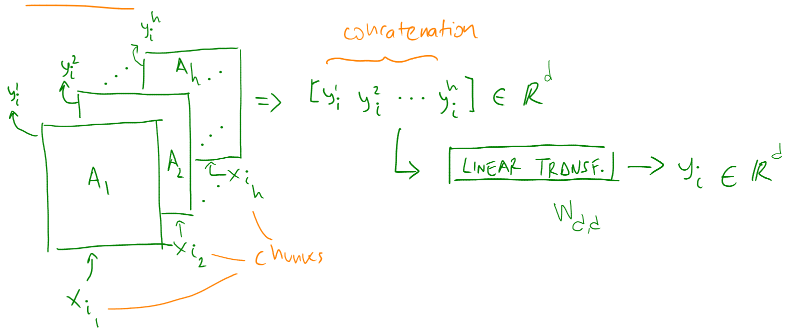
$x_i = [\overbrace{0\,0\,0 \cdots\cdots 0}^{d}]$

INPUT CHUNKS $\to$ Input for A_1 $[0\,0 \cdots 0]$ Input for A_2 $[0\,0 \cdots 0]$ $\cdots$ Input for A_h $[0\,0 \cdots 0] \in \mathbb{R}^{d/h}$

Each set contains 3 matrices. $\leftarrow \underbrace{W_q^1 \; W_k^1 \; W_v^1 \quad W_q^2 \; W_k^2 \; W_v^2 \; \cdots \; W_q^h \; W_k^h \; W_v^h}_{h}$ $W_v \in \mathbb{R}^{d/h \times d/h}$

$\hookrightarrow$ # of Attention heads

Then we have:



$x_i = [x_{i_1}\ x_{i_2} \cdots x_{i_h}]$; $x_i \in \mathbb{R}^d$

CAUSAL MASK

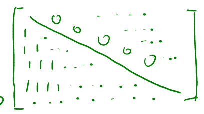
A vanilla multi_head masked self attention layer with a projection at the end. This can also be achieved with torch.nn.MultiheadAttention()

Key, query, value $\in \mathbb{R}^{d \times d}$ where d = embedding size. $\Rightarrow$ k, q, v Projections for all heads: nn.Linear(d, d) →256

Regularization layers → Attention dropout is nn.Dropout(attention_dropout) → 0.1
↳ Residual connection dropout: nn.Dropout(residual_dropout)

Output projection → nn.Linear(d, d)

Mask to hide "future" input for attention (right part) →

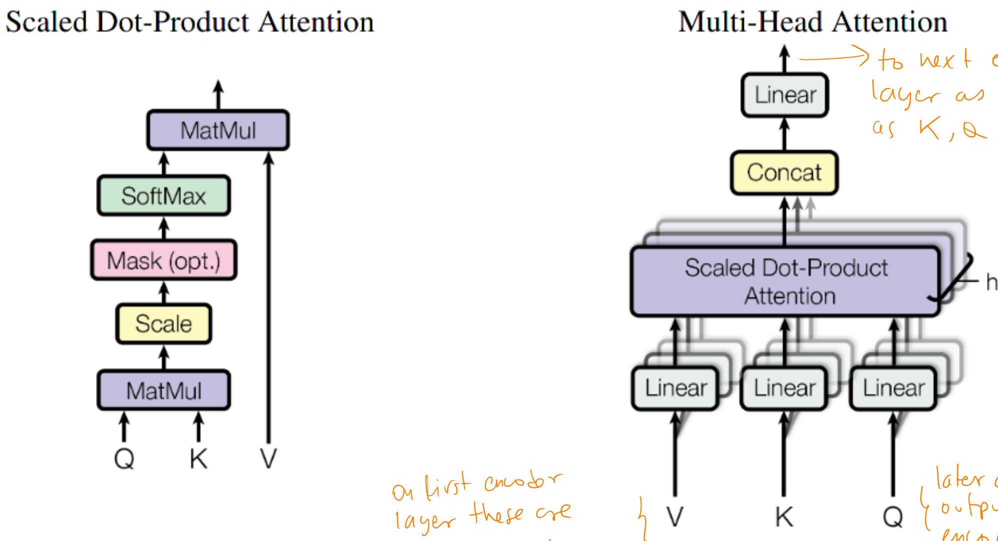


block_size (bs) ← 128

block_size (bs)

But we need a higher dimension tensor of shape (1, 1, bs, bs) with the view function

n_head ⇒ 8



→ to next encoder layer as input as K, Q, V

on first encoder layer these are X

later are the output of prev encoder layer

CS224N – ASSIGNMENT 5

1. Attention exploration (21 points)

(a) (2 points) **Copying in attention:** Recall that attention can be viewed as an operation on a query $q \in \mathbb{R}^d$, a set of value vectors $\{v_1, \ldots, v_n\}, v_i \in \mathbb{R}^d$, and a set of key vectors $\{k_1, \ldots, k_n\}, k_i \in \mathbb{R}^d$, specified as follows:

$$c = \sum_{i=1}^{n} v_i \alpha_i \tag{1}$$

$$\alpha_i = \frac{\exp(k_i^\top q)}{\sum_{j=1}^{n} \exp(k_j^\top q)}. \tag{2}$$

→ attention output (vector)

α_i is attention distribution (scalar)

at time step t

where α_i are frequently called the "attention weights", and the output $c \in \mathbb{R}^d$ is a correspondingly weighted average over the value vectors.

We'll first show that it's particularly simple for attention to "copy" a value vector to the output c. Describe (in one sentence) what properties of the inputs to the attention operation would result in the output c being approximately equal to v_j for some $j \in \{1, \ldots, n\}$. Specifically, what must be true about the query q, the values $\{v_1, \ldots, v_n\}$ and/or the keys $\{k_1, \ldots, k_n\}$?

$q_t \in \mathbb{R}^d$, $K_i \in \mathbb{R}^d$, $v_i \in \mathbb{R}^d$

$i \in \{1 \ldots n\}$ → # hidden cells in encoder

All calculations are done at each time step for all i.

$\frac{X}{C} \Rightarrow \frac{X}{C}$

Caso A → num es muy pequeño ⇒ $\alpha_i \sim 0$

Caso B → num es muy grande ⇒ $\alpha_i \gg 1$

$\frac{X}{C} \Rightarrow \frac{X}{C} \sim 0$

num = $k_i^T q$; $k_i^T = [\underbrace{0 \ 0 \cdots \ 0}_{d}]$ $q = \left[\begin{matrix} \Delta \\ \Delta \\ \vdots \\ \Delta \end{matrix}\right] \} d$

num = $\exp(\underbrace{0\Delta + 0\Delta_2 + \cdots 0\Delta_d}_{\ll 0})$ ⇒ num es muy pequeño

Esto significa q y k o q deben ser vectores con componentes negativos

$k_i \uparrow$ and $q \downarrow$ ⇒ $k_i^T q$ is negative

$[-1, -2, -3] \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = -14$ { this is the extreme case where $k_i = -q$, the absolute opposite

Mientras k_i and q sean más opuestos ⇒ num $\ll 0$ ⇒ $\alpha_i \sim 0$

softmax($k_i^T q$)

n = 4

0 1 2 3

$\alpha_0 \sim 0$ $\alpha_1 \sim 0$ $\alpha_2 \sim 1$ $\alpha_3 \sim 0$

⇒ $c = \cancel{v_0 \alpha_0} + \cancel{v_1 \alpha_1} + v_2 \alpha_2 + \cancel{v_3 \alpha_3}$

⇒ $c \sim v_2$

The attention output will be a copy of one of the inputs when the all the keys except one of them oppose to the query.

$k_j^T q < 0$; $j \neq i$ → Keys k_j opposes to the query (no affinity between the query and keys)

AND $k_i^T q \gg k_j^T q$ → key k_i is less opposed to the query (more affinity between q and k_i)

(b) (4 points) **An average of two:** Consider a set of key vectors $\{k_1, \ldots, k_n\}$ where all key vectors are perpendicular, that is $k_i \perp k_j$ for all $i \neq j$. Let $\|k_i\| = 1$ for all i. Let $\{v_1, \ldots, v_n\}$ be a set of arbitrary value vectors. Let $v_a, v_b \in \{v_1, \ldots, v_n\}$ be two of the value vectors. Give an expression for a query vector q such that the output c is approximately equal to the average of v_a and v_b, that is, $\frac{1}{2}(v_a + v_b)$.[1] Note that you can reference the corresponding key vector of v_a and v_b as k_a and k_b.

$\{k_1, k_2, k_3\}$ when $n = 3$

$k_1 \perp k_2 \Rightarrow k_1 \cdot k_2 = 0$ $\quad k_1 \perp k_3 \quad k_2 \perp k_3$

$\{v_1, v_2, v_3\}$ $\quad \|k_1\| = \|k_2\| = \|k_3\| = 1$

$k_1^T = [1, 0, 0]$
$k_2^T = [0, 1, 0]$
$k_3^T = [0, 0, 1]$

$v_{a=1}, v_{b=3}$

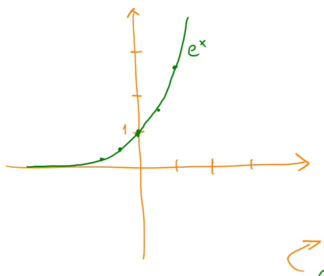
$q = ?$ such as $\boxed{c \approx \frac{v_a + v_b}{2}}$

$\vec{p} = p_1 \vec{i} + p_2 \vec{j} + p_3 \vec{k}$

$\|\vec{p}\| = \sqrt{p_1^2 + p_2^2 + p_3^2}$

To make this happen we need that
$\alpha_a = \alpha_b \approx \frac{1}{2}$, and $\alpha_i \approx 0$ for $i \notin \{a, b\}$ <no affinity between q and k_i!>
$\Rightarrow \text{softmax}(k_a^T q) = \text{softmax}(k_b^T q) \approx \frac{1}{2}$
$\Rightarrow k_a^T q = k_b^T q$

[1]Hint: while the softmax function will never *exactly* average the two vectors, you can get close by using a large scalar multiple in the expression.



Taking the hint into account: Multiple of what? I guess the key vectors since q interacts with them.
$\Rightarrow q = s(k_a + k_b)$
where s is a big scalar

Now, $\alpha_i = \text{softmax}(k_i^T q) = \dfrac{e^{k_i^T q}}{\sum e^{k_j^T q}} \Rightarrow \dfrac{e^{k_i^T s(k_a + k_b)}}{\sum_{i=1}^{n} e^{k_i^T s(k_a + k_b)}} = \dfrac{e^{s k_a k_i^T + s k_b k_i^T}}{\sum_{j=1}^{n} e^{s k_a k_j^T + s k_b k_j^T}}$

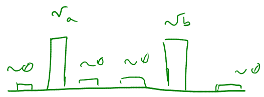
$j \in \{a,b\}$: e^s $\qquad$ $j \notin \{a,b\}$: 1

because the key vectors are orthogonal then if $j \notin \{a,b\}$ $s k_i k_j^T = 0$ for $i \in \{a,b\}$ and $e^0 = 1$

because $k_a k_a^T = 1$ and $k_b k_a^T = 0$

For $i \in \{a,b\}$ $\quad \alpha_a = \dfrac{e^s}{\underbrace{2e^s}_{j \in \{a,b\}} + \underbrace{(n-2) \cdot 1}_{j \notin \{a,b\}}} = \alpha_b \approx 0.5 = 0.5$ if $s = \infty$

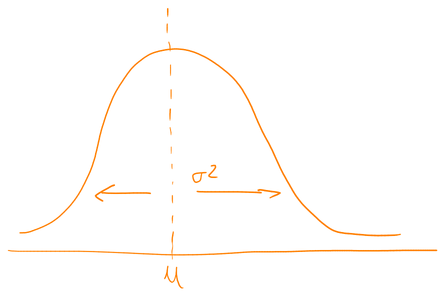
$c \approx$



For $i \notin \{a,b\}$ $\alpha_i \approx 0$ if there is no affinity between the query and the keys to the values for i in these cases. This means q and k are opposite

(c) (5 points) **Drawbacks of single-headed attention:** In the previous part, we saw how it was *possible* for a single-headed attention to focus equally on two values. The same concept could easily be extended to any subset of values. In this question we'll see why it's not a *practical* solution. Consider a set of key vectors $\{k_1, \ldots, k_n\}$ that are now randomly sampled, $k_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, where the means μ_i are known to you, but the covariances Σ_i are unknown. Further, assume that the means μ_i are all perpendicular; $\mu_i^\top \mu_j = 0$ if $i \neq j$, and unit norm, $\|\mu_i\| = 1$.

i. (2 points) Assume that the covariance matrices are $\Sigma_i = \alpha I$, for vanishingly small α. Design a query q in terms of the μ_i such that as before, $c \approx \frac{1}{2}(v_a + v_b)$, and provide a brief argument as to why it works.



$\Sigma_i = \begin{bmatrix} \alpha & 0 & 0 & \cdots & 0 \\ 0 & \alpha & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \alpha \end{bmatrix}$ vanishing small $\alpha \Rightarrow$ The variance $\approx 0 \Rightarrow k_i \approx \mu_i$ $\therefore$ $q = s(\mu_a + \mu_b)$

where s is a big scalar.

ii. (3 points) Though single-headed attention is resistant to small perturbations in the keys, some types of larger perturbations may pose a bigger issue. Specifically, in some cases, one key vector k_a may be larger or smaller in norm than the others, while still pointing in the same direction as μ_a. As an example, let us consider a covariance for item a as $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^\top)$ for vanishingly small α (as shown in figure 1). Further, let $\Sigma_i = \alpha I$ for all $i \neq a$.
When you sample $\{k_1, \ldots, k_n\}$ multiple times, and use the q vector that you defined in part i., what qualitatively do you expect the vector c will look like for different samples?

$\mu_a \mu_a^T = 1$

Not sure about this but I'd say sampling with q as defined in the part i will $\rightarrow$ output big results comparing with the cases where $i \neq a$. Why? because the variance Σ_a is greater than Σ_i. Then the c vector will replicate the large perturbations observed in the keys.

(d) (3 points) **Benefits of multi-headed attention:** Now we'll see some of the power of multi-headed attention. We'll consider a simple version of multi-headed attention which is identical to single-headed self-attention as we've presented it in this homework, except two query vectors (q_1 and q_2) are defined, which leads to a pair of vectors (c_1 and c_2), each the output of single-headed attention given its respective query vector. The final output of the multi-headed attention is their average, $\frac{1}{2}(c_1+c_2)$. As in question 1(c), consider a set of key vectors $\{k_1, \dots, k_n\}$ that are randomly sampled,

$k_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, where the means μ_i are known to you, but the covariances Σ_i are unknown. Also as before, assume that the means μ_i are mutually orthogonal; $\mu_i^\top \mu_j = 0$ if $i \neq j$, and unit norm, $\|\mu_i\| = 1$.

i. (1 point) Assume that the covariance matrices are $\Sigma_i = \alpha I$, for vanishingly small α. Design q_1 and q_2 such that c is approximately equal to $\frac{1}{2}(v_a + v_b)$.

For this to happen $c_1 = v_a$ and $c_2 = v_b$ and as per question a) that will be true if there is equal affinity between the query and the keys k_a and k_b, and no affinity with k_i for $i \notin \{a, b\}$.

Now taking question b) into account we can say that $q_1 \approx S k_a$ and $q_2 \approx S k_b$. Finally, as $\Sigma_i = \alpha I$ for a vanishing small α we know that the variance $\sigma_i^2 \approx 0$ $\therefore$

$$q_1 = S\mu_a \,, \quad \text{where } S \text{ is a big scaler.}$$
$$q_2 = S\mu_b$$

ii. (2 points) Assume that the covariance matrices are $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^\top)$ for vanishingly small α, and $\Sigma_i = \alpha I$ for all $i \neq a$. Take the query vectors q_1 and q_2 that you designed in part i. What, qualitatively, do you expect the output c to look like across different samples of the key vectors? Please briefly explain why. You can ignore cases in which $q_i^\top k_a < 0$.

$c = \frac{c_1 + c_2}{2}$; $\sigma_a^2 > \sigma_{i \neq a}^2$

If we have the same q_1 and q_2 then c should be $\approx (v_a + v_b)/2$ then when sampling keys there would be a tendency of having bigger norm for the c vector around v_a, but as we are averaging c_1 and c_2 the perturbations might be reduced.

(e) (7 points) **Key-Query-Value self-attention in neural networks:** So far, we've discussed attention as a function on a set of key vectors, a set of value vectors, and a query vector. In Transformers, we perform *self-attention*, which roughly means that we draw the keys, values, and queries from the same data. More precisely, let $\{x_1, \dots, x_n\}$ be a sequence of vectors in $\mathbb{R}^d$. Think of each x_i as representing word i in a sentence. One form of self-attention defines keys, queries, and values as follows. Let $V, K, Q \in \mathbb{R}^{d \times d}$ be parameter matrices. Then

$$v_i = Vx_i \quad i \in \{1, \dots, n\} \tag{3}$$
$$k_i = Kx_i \quad i \in \{1, \dots, n\} \tag{4}$$
$$q_i = Qx_i \quad i \in \{1, \dots, n\} \tag{5}$$

Then we get a context vector for each input i; we have $c_i = \sum_{j=1}^{n} \alpha_{ij} v_j$, where α_{ij} is defined as $\alpha_{ij} = \frac{\exp(k_j^\top q_i)}{\sum_{\ell=1}^{n} \exp(k_\ell^\top q_i)}$. Note that this is single-headed self-attention.
In this question, we'll show how key-value-query attention like this allows the network to use different aspects of the input vectors x_i in how it defines keys, queries, and values. Intuitively, this allows networks to choose different aspects of x_i to be the "content" (value vector) versus what it uses to determine "where to look" for content (keys and queries.)

$c_2 = \sum_{j=1}^{n} \alpha_{2j} v_j$ where

$\alpha_{2j} = \text{Softmax}(k_j^T q_2)$

↳ the keys showing more affinity with q_2

A vector highlighting the characteristics of the input values with higher affinity w.r.t. q_2

i. (3 points) First, consider if we didn't have key-query-value attention. For keys, queries, and values we'll just use x_i; that is, $v_i = q_i = k_i = x_i$. We'll consider a specific set of x_i. In particular, let u_a, u_b, u_c, u_d be mutually orthogonal vectors in $\mathbb{R}^d$, each with equal norm $\|u_a\| = \|u_b\| = \|u_c\| = \|u_d\| = \beta$, where β is very large. Now, let our x_i be:

$$x_1 = u_d + u_b \tag{6}$$
$$x_2 = u_a \tag{7}$$
$$x_3 = u_c + u_b \tag{8}$$

If we perform self-attention with these vectors, what vector does c_2 approximate? Would it be possible for c_2 to approximate u_b by adding either u_d or u_c to x_2? Explain why or why not (either math or English is fine).

Since $v_i = q_i = k_i = x_i$ then

$$c_2 = \sum_{j=1}^{n} \text{softmax}(x_j^T x_2) x_j = \frac{x_1 e^{x_1^T x_2} + x_2 e^{x_2^T x_2} + x_3 e^{x_3^T x_2}}{e^{x_1^T x_2} + e^{x_2^T x_2} + e^{x_3^T x_2}}$$

$\Downarrow$

$$c_2 = x_1 \,\text{softmax}(x_1^T x_2) + x_2 \,\text{Softmax}(x_2^T x_2) + x_3 \,\text{Softmax}(x_3^T x_2)$$

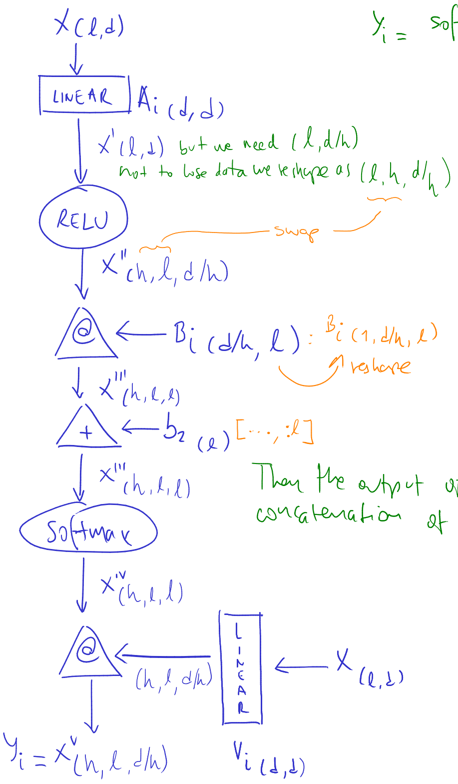
Let's take a look to $x_1^T x_2$. $x_1^T x_2 = \underbrace{(u_d + u_b)^T u_a}_{u_a u_d^T + u_a u_b^T}$. Since the dot product is distributive with addition:

As u_a, u_b and u_d are orthogonal among them the dot product is 0. Same happens with $x_3^T x_2$

Then we have: $c_2 \approx x_2 \,\text{softmax}(x_2^T x_2) \approx \frac{x_2 e^{x_2^T x_2}}{e^{x_1^T x_2} + e^{x_2^T x_2} + e^{x_3^T x_2}} \approx x_2 \frac{e^{x_2^T x}}{2 + e^{x_2^T x_2}} \approx x_2 \approx u_a$ $\#$

<u>Notice</u>: $\text{softmax}(x_1^T x_2) = \frac{e^0}{2 + e^{x_2^T x_2}} = \frac{1}{2 + e^{\beta^2}} \approx 0$

Part g) Synthesizer Attention



Linear transf. with W_1

W_2 in the implementation code. Not a linear transf.

The same in the implementation code

$$Y_i = \text{softmax}(\text{ReLU}(XA_i + b_1)B_i + b_2)(XV_i)$$ → A linear transformation with self.value()

$$\text{Softmax}(\underbrace{\underbrace{\underbrace{l\times d \quad d\times d}_{l\times d = l\times h\times d/h} \quad \underbrace{d/h\times l}_{d/h \times l}}_{l\times l}}_{l\times l}) \quad \underbrace{l\times d \quad d\times d}_{l\times d = l\times h\times d/h \;\to\; l\times d/h}$$

$$\underbrace{l\times l \quad l \times d/h}_{l\times d/h}$$

Then the output of the synthesizer mechanism is a liner transformation of the concatenation of each Attention head output:

$$Y = [Y_1, Y_2 \dots Y_h]W$$

W ↳ $d\times d$

$\underbrace{[Y_1, Y_2 \dots Y_h]}_{}$: $l\times \frac{d}{h}$ times h => $l\times d$

Y: $l\times d$ ↳ Same shape as the input X!

ii. (2 points) Why might the *synthesizer* self-attention not be able to do, in a single layer, what the key-query-value self-attention can do?

I'd say it is because the synthesizer is not inspecting the potential relationships between each possible combination in the input. It is basically transforming the input into another representation and convering it with the value representation. It lacks the power of the Q, K, V system.

3. Considerations in pretrained knowledge (5 points)

(a) (1 point) Succinctly explain why the pretrained (vanilla) model was able to achieve an accuracy of above 10%, whereas the non-pretrained model was not.

Because the Vanilla model w/o pretraining was trained as a language model without an "understanding" of the complex relationships in the language (person $\xrightarrow{\text{born}}$ place). The pretraining activity was able to capture some of that language dependencies boosting the final objective thanks to transfer learning. Also the pretraining task had better chances to learn helpful dependencies for the downstream tasks because its dataset was bigger (68%) than the finetuning dataset

(b) (2 points) Take a look at some of the correct predictions of the pretrain+finetuned vanilla model, as well as some of the errors. We think you'll find that it's impossible to tell, just looking at the output, whether the model *retrieved* the correct birth place, or *made up* an incorrect birth place. Consider the implications of this for user-facing systems that involve pretrained NLP components. Come up with two reasons why this indeterminacy of model behavior may cause concern for such applications.

I think the model has learned to generate a place name after a question like "Where was xxx born?" When it gives a correct answer is probably because a relationship between that person name and a place was captured and embedded during pretraining. Otherwise outputs any location name. The implications are big, focus on trust and confidence of the predictions.

Low confidence → can't be applied to critical processes

Poor prediction → Low usability rate. Undesirable consequences when acting blindly based on erroneus model output.

(c) (2 points) If your model didn't see a person's name at pretraining time, and that person was not seen at fine-tuning time either, it is not possible for it to have "learned" where they lived. Yet, your model will produce *something* as a predicted birth place for that person's name if asked. Concisely describe a strategy your model might take for predicting a birth place for that person's name, and one reason why this should cause concern for the use of such applications.

If the prediction score is below a threshold the model can query a database to retrieve the real answer. This of course defeats the purpose of the model and might wrongly bias the user about the real power of AI. Perhaps it is better to take advantage of this situation to interact with the user asking for the missing info and/or creating a new datapair from the database for later finetuning.