Setting

logit of class k $\{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$

- **Goal:** predicting $y \in \mathbb{R}$ from input $x \in \mathbb{R}^d$
- assume that we are given training data

$$\{(x^{(1)}, y^{(1)}, c^{(1)}), (x^{(2)}, y^{(2)}, c^{(2)}), \dots, (x^{(n)}, y^{(n)}, c^{(n)})\},$$

  where $X^{(i)}, y^{(i)}$ are as usual, and $c^{(i)} \in \mathbb{R}^k$ are concept vectors
- **Example:** concepts from the arthritis task: `sclerosis`, `bone spurs`, ...
- $c^{(i)} = (10, 0.1, -0.3, \dots)^\top$ corresponds to `sclerosis` being present
- **Concept bottleneck model:** $f(x) = g(h(x))$, where
  - $h : \mathbb{R}^d \to \mathbb{R}^k$ (K) predicts concepts from input
  - $g : \mathbb{R}^k \to \mathbb{R}$ (K) predicts output from concepts

$$f(x) = g(h(x))$$

Independent bottleneck

- there are several natural ways to train $f = g \circ h$
- let us call $\hat{g}$ and $\hat{h}$ the trained versions of g and h
- **Loss functions:**
  - $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$ for the outputs
  - $\forall j \in [k]$, define $\ell_j : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$ the loss for concept j

$\left(\ell_j = -\log(c_j)\right)$

1) **Independent bottleneck:** learn $\hat{g}$ and $\hat{h}$ independently:

$$\hat{h} \in \arg\min_h \sum_{i=1}^{n} \sum_{j=1}^{k} \ell_j(h_j(x^{(i)}), c_j^{(i)}) \,, \qquad \text{and} \qquad \hat{g} \in \arg\min_g \sum_{i=1}^{n} \ell(g(c^{(i)}), y^{(i)}) \,.$$

- **Intuition:** learn (independently) a good concept predictor and a good predictor relying only on concepts
- **Beware:** although $\hat{g}$ trained using true concepts, $\hat{f} = \hat{g} \circ \hat{h}$

Other possibilities

+ from prev. slide

2) ▶ **Sequential bottleneck:** $\hat{h}$ learned as before, $\hat{g}$ learned using $\hat{h}$

▶ namely,

$$\hat{h} \in \arg\min_h \sum_{i=1}^{n} \sum_{j=1}^{k} \ell_j(h_j(x^{(i)}), c_j^{(i)}), \qquad \text{and} \qquad \hat{g} \in \arg\min_g \sum_{i=1}^{n} \ell(g(\hat{h}(x^{(i)})), y^{(i)}).$$

3) ▶ **Joint bottleneck:** minimize a weighted sum of the two objectives:

$$(\hat{g}, \hat{h}) \in \arg\min_{g,h} \sum_{i=1}^{n} \left[\underbrace{\ell(g(h(x^{(i)})), y^{(i)})}_{f(x^{(i)}),\, y^{(i)}} + \lambda \sum_{j=1}^{k} \ell_j(h_j(x^{(i)}), c_j^{(i)}) \right],$$

pred. of network

with $\lambda > 0$ some hyperparameter

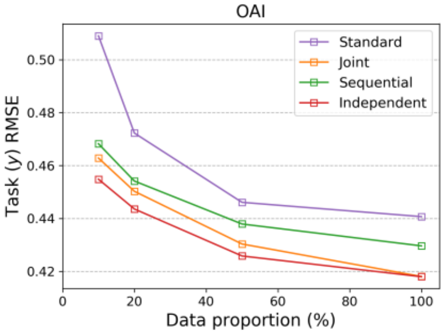
4) ▶ **Standard model:** ignores concepts altogether:

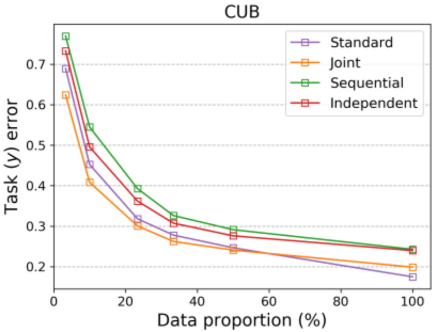
Do not need $c^{(i)}$

$$\hat{g}, \hat{h} \in \arg\min_{g,h} \sum_{i=1}^{n} \ell(\underbrace{g(h(x^{(i)}))}_{f(x^{(i)})}, y^{(i)}).$$

Empirical results

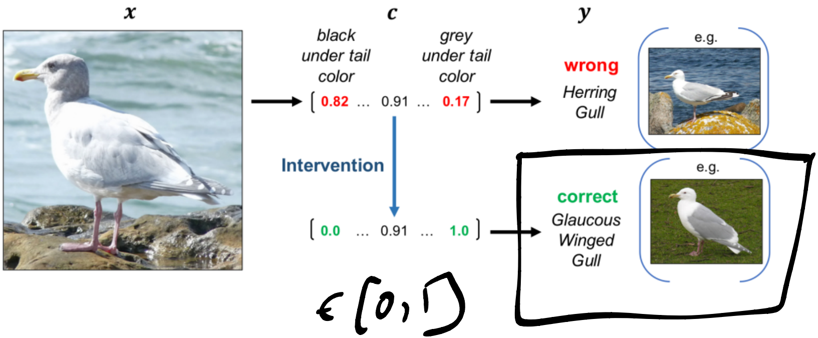
- all models are good at predicting concepts
- then the metric is really accuracy: depends on the task
- ... and always a bit smaller than without relying on concepts :(





Concept intervention

- **Concept intervention:** modifying concept values to get more accurate prediction
- **Example:**



Summary

- **Concept bottleneck:** explainable-by-design concept-based model
- requires user-defined concepts
- allows for concept intervention
- many extensions
- **Remark:** also possible to perform transplantation on existing network, introducing concept layer instead of existing layer

9. Explanations for attention-based models

9.1. Attention mechanism

Setting

- in this chapter, we work in the context of NLP $\rightarrow$ same context as for *LIME for text data*
- **Reminder:** document x = ordered sequence of T tokens $\longrightarrow$ $x = (x_1, x_2, x_3, \ldots, x_T)$
- for simplicity's sake, dictionary $= [D]$ with $D \geq 1$
- **Example:** $D = 10$

$$x = (5, 0, 3, 3, 7, 9)\,.$$

- T = length of the document is 6
- tasks we have in mind:
  - *classification:* given x, predict the correct class (example: sentiment analysis)
  - *next-token prediction*, a.k.a. sequence modeling: given $(x_1, x_2, \ldots, x_{t-1})$, make a reasonable guess for x_t (example: language modeling)
  - *sequence-to-sequence:* given x, predict another sequence y (example: neural machine translation)
- attention is a mechanism used in modern architectures to tackle those

$\hat{d_e} = 10^2$

Token embeddings

Bahdanau et al, '15
Vaswani et al, '17

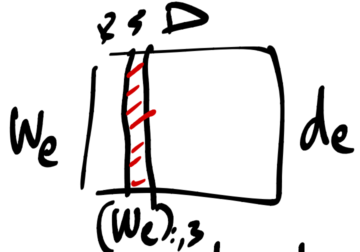
- **First step:**[93] vector representation of each token
- for each $t \in [T]$, token $x_t = j$ is embedded as

$\in \{1, \ldots, D\}$

$$e_t := (W_e)_{:,j} + W_p(t) \in \mathbb{R}^{d_e}\,,$$

$(\ldots, \underline{3}, \ldots, \underline{3},)$

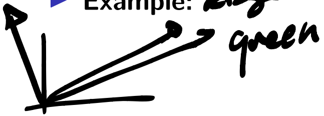
where:
  - $W_e \in \mathbb{R}^{d_e \times D}$ matrix containing embeddings of all tokens
  - $W_p : \mathbb{N} \to \mathbb{R}^{d_e}$ positional embedding



d_e = dim embedding

lin

- **Typically,** W_e and W_p are learned, W_p can also be set to something arbitrary
- **Example:** king



$$\begin{cases} W_p(t)_{2i} & = \cos(t/T_{\max}^{2i/d_e}) \\ W_p(t)_{2i-1} & = \sin(t/T_{\max}^{2i/d_e})\,. \end{cases}$$

[93] I am following Phuong and Hutter, *Formal Algorithms for Transformers*, preprint, 2022

Keys, queries, values

$x \mapsto E = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \end{pmatrix}$ T, d_e

- max length for documents $= T_{\max}$
- **Note:**[94] in modern architectures, $T_{\max} \approx 10^5$
- **Padding** until $T_{\max}$ with <EOS> token, to simplify $T = T_{\max}$ in these notes
- **Next step:** for each $t \in [t]$, e_t transformed into:

$$\begin{cases} k_t := W_k e_t + b_k \in \mathbb{R}^{d_{\text{att}}} & \text{(key)} \\ q_t := W_q e_t + b_q \in \mathbb{R}^{d_{\text{att}}} & \text{(query)} \\ v_t := W_v e_t + b_v \in \mathbb{R}^{d_{\text{out}}} & \text{(value)} \end{cases}$$

d_{att} = dim. attention

- matrices $W_k, W_q \in \mathbb{R}^{d_{\text{att}} \times d_e}$, and $W_v \in \mathbb{R}^{d_{\text{out}} \times d_e}$ are learned
- bias vectors $b_k, b_q \in \mathbb{R}^{d_{\text{att}}}$, $b_v \in \mathbb{R}^{d_{\text{out}}}$ also learnable, set to zero for simplicity

[94] for instance, Claude 2.1 has a context size of 200k tokens, corresponding to roughly the length of Bram Stoker's *Dracula*

Single-query attention

$$\begin{pmatrix} 0.1 \\ -0.2 \\ \vdots \\ 5 \end{pmatrix} \longmapsto \begin{pmatrix} 0.01 \\ 0.01 \\ \vdots \\ 0.2 \end{pmatrix}$$

- **Definition:** for any vector $u \in \mathbb{R}^d$, define coordinate-wise

$$\forall j \in [d], \quad \boxed{\operatorname{softmax}(u)_j = \frac{e^{u_j}}{\sum_k e^{u_k}}} \, . \quad \in \mathbb{R}^d \quad \in [0,1]^d$$

- **Intuition:** squeezes everyone into $[0,1]$; if coordinate j much higher then close to 1
- for a given query $q \in \mathbb{R}^{d_{\text{att}}}$, each token x_t with $t \in [T_{\max}]$ receives *attention weight* [95]

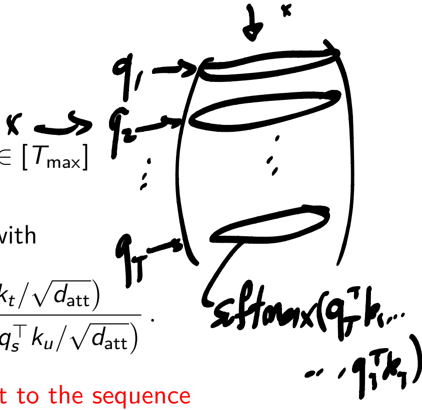
q_s

$$\alpha_t = \operatorname{softmax}(q^\top k_1, \ldots, q^\top k_{T_{\max}}) = \frac{\exp\left(q^\top k_t / \sqrt{d_{\text{att}}}\right)}{\sum_{u=1}^{T_{\max}} \exp\left(q^\top k_u / \sqrt{d_{\text{att}}}\right)} \, .$$

- **Intuition:** if query "matches" with k_t, then α_t large

[95] Bahdanau, Cho, Bengio, *Neural machine translation by jointly learning to align and translate*, ICLR, 2015

Self-attention



- **Typical situation:** compute attention for $q = q_t$, for all $t \in [T_{\max}]$
- this is called self-attention
- formally speaking, compute the matrix $A(x) \in \mathbb{R}^{T_{\max} \times T_{\max}}$ with

$$\forall s, t \in [T_{\max}], \qquad A(x)_{s,t} = \frac{\exp\left(q_s^\top k_t / \sqrt{d_{\text{att}}}\right)}{\sum_{u=1}^{T_{\max}} \exp\left(q_s^\top k_u / \sqrt{d_{\text{att}}}\right)} .$$

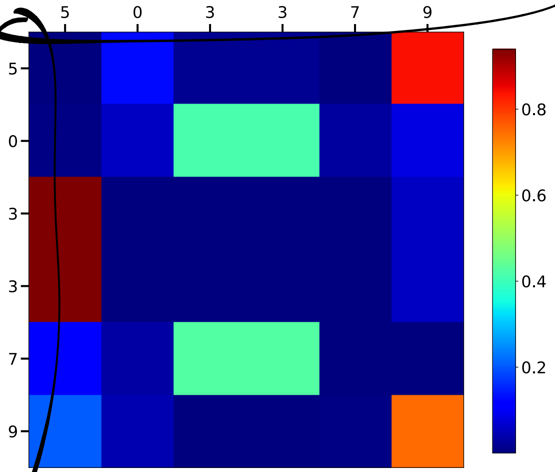
- rows of $A(x)$ correspond to attention of tokens with respect to the sequence
- **Additional notation:** $E = E(x) \in \mathbb{R}^{T \times d_e}$ collection of embeddings
- $Q = EW_q^\top \in \mathbb{R}^{T \times d_{\text{att}}}$, $K = EW_k^\top \in \mathbb{R}^{T \times d_{\text{att}}}$
- extending definition of softmax to matrices (row-wise):

$Q(x)$ $K(x)$

$$\boxed{A = \text{softmax}(QK^\top / \sqrt{d_{\text{att}}}) .}$$

Self-attention, example

- **Example:** computing self-attention for the example sequence



$q_t^T k_t$

• ~~sim:~~ $\cancel{q_t^T q_t = \|q_t\|^2 > 0}$

• orth: $q_t^T k_t \simeq 0$

Values

$A(x)$ $\to \tilde{v}_s$

rows of A

- **Recall:** values

$$\forall t \in [T_{\max}], \qquad v_t = W_v e_t \in \mathbb{R}^{d_{\text{out}}} .$$

- **Final step:** aggregate value vectors depending on attention coefficients
- namely, for all $s \in [T_{\max}]$,

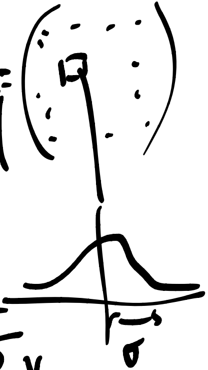
$$\boxed{\tilde{v}_s = \sum_{t=1}^{T_{\max}} A(x)_{s,t} v_t \in \mathbb{R}^{d_{\text{out}}} .}$$

- **To summarize:** attention blocks take as input sequence of T tokens and outputs T vectors of size d_{out}
- **Intuition:** key $=$ description, query $=$ what we are looking for
- value $=$ convex combination of the values with weight close to 1 if e_s and e_t match

"to match": $q^T k >> 0$

More intuition

$W_q =$



- **At initialization:** W_q and W_k random matrices (coef. i.i.d. $\mathcal{N}(0, \sigma^2)$)
- thus q_s and k_t are orthogonal with high probability and

$$\forall s, t \in [T_{\max}], \qquad \underline{q_s^\top k_t \approx 0\,.}$$

- the attention scores look like

$$\left(\mathrm{softmax}(0, \ldots, 0)\right)_j = \frac{e^0}{\sum e^0} = \frac{1}{T}$$

$$A(x) \approx \begin{pmatrix} 1/T & 1/T & \cdots & 1/T \\ 1/T & 1/T & \cdots & 1/T \\ \vdots & & & \vdots \\ 1/T & 1/T & \cdots & 1/T \end{pmatrix} \qquad \frac{1}{T}\sum_{t=1}^{T} v_t$$

- initial value vector = average
- progressively learn to put more weights on some tokens depending on the task we are training for

Masked self-attention

- **Masked self-attention:** remove the corresponding $q_s^\top k_t$ from the softmax computation
- trick $=$ define a mask with $-\infty$ when we want to ignore (and 1 otherwise)
- then multiply element-wise the $QK^\top$ matrix
- **Important example:** constrain the model to ignore "future" tokens
- namely, use only $x_1, \ldots, x_{t-1}$ to predict x_t *(unidirectional attention)*
- define $M_{s,t} = -\infty$ if $s \leq t$, 1 otherwise
- masked self-attention is given by

$$A(x, M) = \mathrm{softmax}((M + QK^\top)/\sqrt{d_{\mathrm{att}}})\,.$$

$\cancel{q_1^\top k_5}$

'looking into the future'

- **Why?** on a given line, $\mathrm{e}^{q_s^\top k_t} = \mathrm{e}^{-\infty} = 0$ whenever $s > t$, meaning that

$$\forall s > t, \quad A(x, M)_{s,t} = \frac{\mathrm{e}^{q_s^\top k_t}}{\sum_{u=1}^{s} \mathrm{e}^{q_s^\top k_u}}, \quad \text{and 0 otherwise.}$$

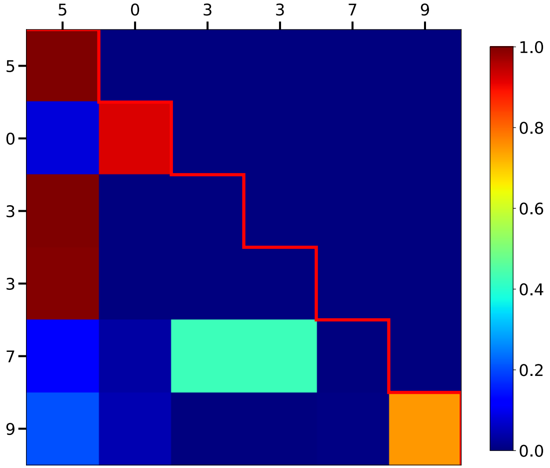
$e^{q_s^\top k_t - \infty} = 0$

$\Sigma \ldots$

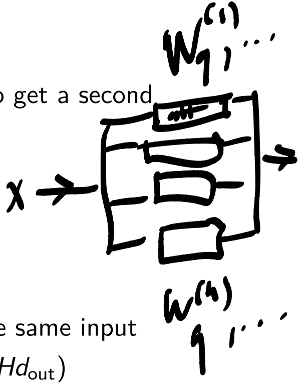
$QK^T + M$

Masked self-attention, example

- **Example:** computing masked self-attention for the example sequence



Further refinements



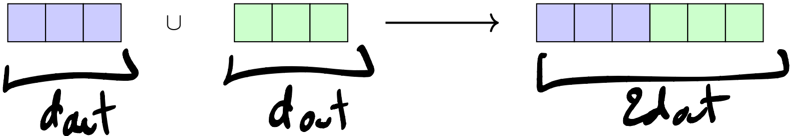
- **Cross-attention:** in the context of sequence-to-sequence, typical to get a second sequence as context
- namely, take $Q = Q(x)$ and $K = K(z)$, then compute

$$A(x,z) = \mathrm{softmax}(QK^\top / \sqrt{d_{\mathrm{att}}})$$

as before

- **Multi-head:** usually, several attention blocks work in parallel on the same input
- say H heads $\rightarrow$ concatenate the H outputs $T \times d_{\mathrm{out}}$ to form $T \times (Hd_{\mathrm{out}})$
- **Illustration:**

$H d_{out} = d_e$



9.2. Transformers: the example of GPT-2

GPT-2

- attention mechanism was popularized by the transformer architecture[96]
- in this section, I give more details about GPT-2-small ($\approx$ 117M)[97]
- **Overview:**
  - BytePair[98] tokenized input $x \in [D]^T$ ($D = 50,304$)
  - embedding as described in previous section ($d_e = 768$) $x \mapsto f^{(0)} \in \mathbb{R}^{T \times d_e}$
  - $L = 12$ sequential unidirectional self-attention layers
  - each layer has 12 heads ($d_{\text{out}} = d_e/12 = 64$) $f^{(t)} \mapsto f^{(t+1)} \in \mathbb{R}^{T \times d_e}$
  - final output: linear transformation and softmax $f^{(L)} \mapsto f(x) \in \mathbb{R}^{T \times D}$

D

$x \longmapsto T\left(0.8 \right) 5$

[96]Vaswani et al., *Attention is all you need*, NeurIPS, 2017
[97]Radford et al., *Language Models are Unsupervised Multitask Learners*, preprint, 2019
[98]Sennrich et al., *Neural machine translation of rare words with subword units*, Proc. ACL, 2016

BytePair encoding

- **Overall idea:** encode rare words by subword units
- **Intuition:** compound words

“Abwasserbehandlungsanlage” $\mapsto$ “Abwasser|behandlungs|anlage”

- adaptation of a compression algorithm[99] to the word segmentation task
- start from tokens = characters
- for a given number of merges:
  1. find the most frequent *token pair* in the dataset
  2. assign a new token to this pair
- **Example:** (’low’,’login’)

A

→ lo ow lo og gi in

→ A-ow ow-A A-og ....

[99]Gage, *A new algorithm for data compression*, C. Users J., 1994

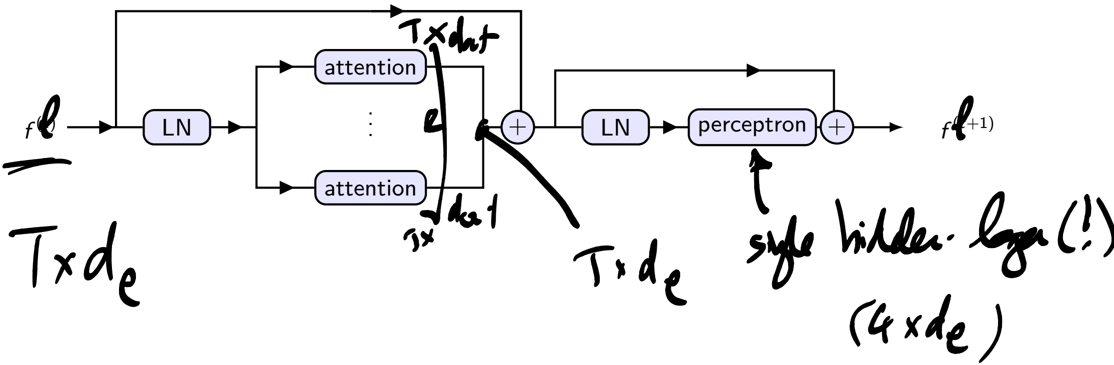
GPT-2 block

- sequentially, input $f^{(t)} \in T \times d_e$ goes through
  - $H = 12$ unidirectional self-attention heads $\rightarrow$ output $\in \mathbb{R}^{T \times d_{\text{out}}}$ with $d_{\text{out}} = d_e/12 = 64$
  - concatenate everyone, back in $\mathbb{R}^{d_e}$
  - single-layer perceptron
    - works on each token representation independently (input $\in \mathbb{R}^{d_e}$)
    - hidden layer of size $4 \times d_e = 3,072$
    - GeLU activation
    - output again in $\mathbb{R}^{d_e}$
  - layer output is $f^{(t+1)} \in \mathbb{R}^{T \times d_e}$
- each attention head works in parallel, but there are some connections
- **Additionally:** layer-norm before and after self-attention, skip connections

GPT-2 block, ctd.

x 12

- **Schematically:**



Layer normalization

- **Layer normalization:** alternative to batch normalization
- **Overall idea:** normalize across all features from a layer[100]
- namely, if layer h has features $f = (f_1, \ldots, f_d)^\top \in \mathbb{R}^d$, set

$$\mu := \frac{1}{d}\sum_{j=1}^{d} f_j \quad \text{and} \quad \sigma^2 := \frac{1}{d}\sum_{j=1}^{d}(f_j - \mu)^2$$

- then

$$\forall j \in [d], \qquad \mathrm{LN}(f) := \gamma_j \frac{f_j - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta_j\,,$$

where ε is a small, positive offset, while γ and β are **learnable parameters**

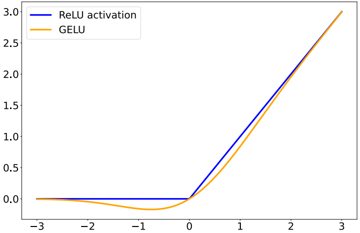
[100] Ba, Kiros, Hinton, *Layer normalization*, preprint, 2016

Gaussian error linear units (GELUs)

- **GeLUs:**[101] smoothed version of ReLU
- **Recall:** Φ is the cumulative distribution function of a $\mathcal{N}(0,1)$:

$$\Phi(x) = \mathbb{P}\left(\mathcal{N}(0,1) \leq x\right) = \frac{1}{2\pi} \int_{-\infty}^{x} \mathrm{e}^{\frac{-t^2}{2}} \mathrm{d}t \, .$$

- then $\mathrm{GELU}(x) := x\Phi(x)$



[101] Hendrycks and Gimpel, *Gaussian error linear units*, preprint, 2016

Querying the model at train time

- after the last attention layer, $f^{(L)} \in \mathbb{R}^{T \times d_e}$
- linear transformation **with same weights as embedding**[102] $f^{(L)} \mapsto f^{(L)} W_e \in \mathbb{R}^{T \times D}$
- then softmax on each row:

$$f(x) = \mathrm{softmax}(f^{(L)} W_e) \in \mathbb{R}^{T \times D}.$$

- for each token, discrete probability distribution on the dictionary = proba of next token
- **At training time:** binary cross entropy between the predictions and the example:

$$\mathrm{loss}(x^{(1)}, \ldots, x^{(n)}) = \sum_{i=1}^{n} \sum_{t \in [T-1]} -\log f(x^{(i)})_{\tilde{x}^{(i)}_{t+1}}.$$

- minimize this loss on `WebText` dataset with Adam[103]

[102] Press and Wolf, *Using the Output Embedding to Improve Language Models*, EACL, 2017
[103] Kingma and Ba, *Adam: A Method for Stochastic Optimization*, ICLR, 2015

Querying the model at test time

- prompt

→ dist on D
⇔ proba of next tok.

- **Decoding:** several options, corresponding to the use-case:
  - *classification:* train regressor on (part of) $f^{(L)}(x)$ features
  - *next-token prediction:* use the prediction from the last row $f(x)_{T,:}$, usually take the argmax
  - *sequence generation:* iterate next-token prediction, stop when generating `<EOS>`
- **Reminder:** $\arg\max u$ = index of the coordinate of u with maximal value
- **Remark:** not necessarily taking the argmax when generating sequence:
  - *pure sampling:* sample according to the proba distribution on $[D]$
  - *top-k sampling:* sample only among the top-k elements of $[D]$
  - *beam search:* sample ahead and maximize product proba
  - *sampling with temperature:*[104] sampling with skewed softmax
  - *nucleus sampling:*[105] adaptive top-k sampling
  - ...

[104] Ackley, Hinton, Sejnowski, *A learning algorithm for Boltzmann machines*, Cognitive Science, 1985
[105] Holtzman et al., *The curious case of neural text degeneration*, ICLR, 2020

9.3. Explaining transformers

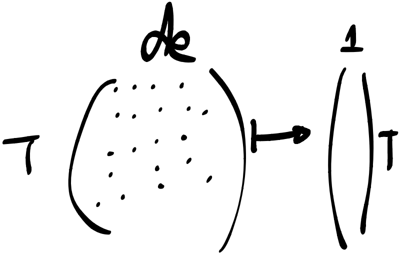
Classification setting

- **Reminder:** in that case, our model takes *real values*
- we can use standard techniques
- **Example:** gradient with respect to the input
- **Problem:** input is a sequence of discrete tokens... (general issue in XAI for NLP)
- **Solution:** decompose model into $f = g \circ e$, where

$$e : [D]^T \longrightarrow \mathbb{R}^{T \times d_e}$$

  embedding function
- compute $\nabla_{e(\xi)} g \in \mathbb{R}^{T \times d_e}$, then **map back to original sequence**
- that is, aggregate the information for each token

Classification setting, ctd.

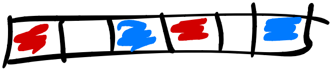


- typical solutions for aggregation:
  - *mean value:*[106]

$$\text{G-avg}_t = \frac{1}{d_e}\sum_{j=1}^{d_e} (\nabla_{e(\xi)} g)_j$$

  - L^1*-norm:*[107]

$$\text{G-L1}_t = \frac{1}{d_e}\sum_{j=1}^{d_e} \left|(\nabla_{e(\xi)} g)_j\right|$$



  - L^2*-norm:*[108]

$$\text{G-L2} = \sqrt{\frac{1}{d_e}\sum_{j=1}^{d_e} \left|(\nabla_{e(\xi)} g)_j\right|^2}$$

  - ...

[106] Atanasova et al., *A diagnostic study of explainibility techniques for text classification*, EMNLP, 2020
[107] Li et al., *Visualizing and understanding models in NLP*, Proc. ACL, 2016
[108] Poerner et al., *Evaluating neural network explanation methods using hybrid documents and morphosyntactic agreement*, Proc. ACL, 2018

Generative setting

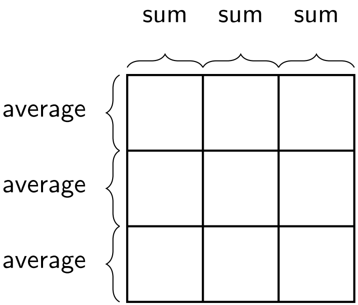
- in that case no clear target...
- **Natural idea:** look directly at the attention scores of self-attention heads
- get insights on what a particular head is doing
- **Problem:** most tokenizers are "sub-words"
- need to transform token-to-token into word-to-word attention map
- **Solution:**[109]
  - for attention *to* a split-up word, *sum* attention weights
  - for attention *from* a split-up word, *average* attention weights
- formally, if s (resp. t) is split into $s_1, \ldots, s_a$ (resp. $t_1, \ldots, t_b$), define

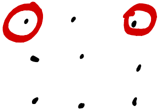
$$\tilde{A}_{s,t} := \frac{1}{a} \sum_{i=1}^{a} \sum_{j=1}^{b} A_{s_i, t_j} \,.$$

[109] Clark et al., *What does BERT look at? An analysis of BERT's attention*, 2nd BlackBoxNLP workshop (ACL), 2019

Proof of the claim

- **Claim:** rows of $\tilde{A}$ still sum to one
- proof with a drawing:





GPT-2

Looking at individual heads: example

- **Example from the paper:** looking at BERT (X-Y stands for head Y in layer X)