Lecture Notes for

Machine Learning in Python

Professor Eric Larson

Final Lecture: Position, Attention, Retrospective

Lecture Agenda

- Logistics
  - Grading Update
  - Sequential Networks due **Last Day of Finals**
    - **Before Midnight on December 14**
  - I will have grades finalized by **late December 15**
- Agenda
  - More efficient Transformers
  - Retrospective and Evaluations

Class Overview, by topic

Table Data Visualization
Numpy, Pandas, Seaborn
Overviews with some in-depth discussion

Dimension Reduction and Image Processing
Scikit-learn, Scikit Image,
Intuition only, Some mathematics

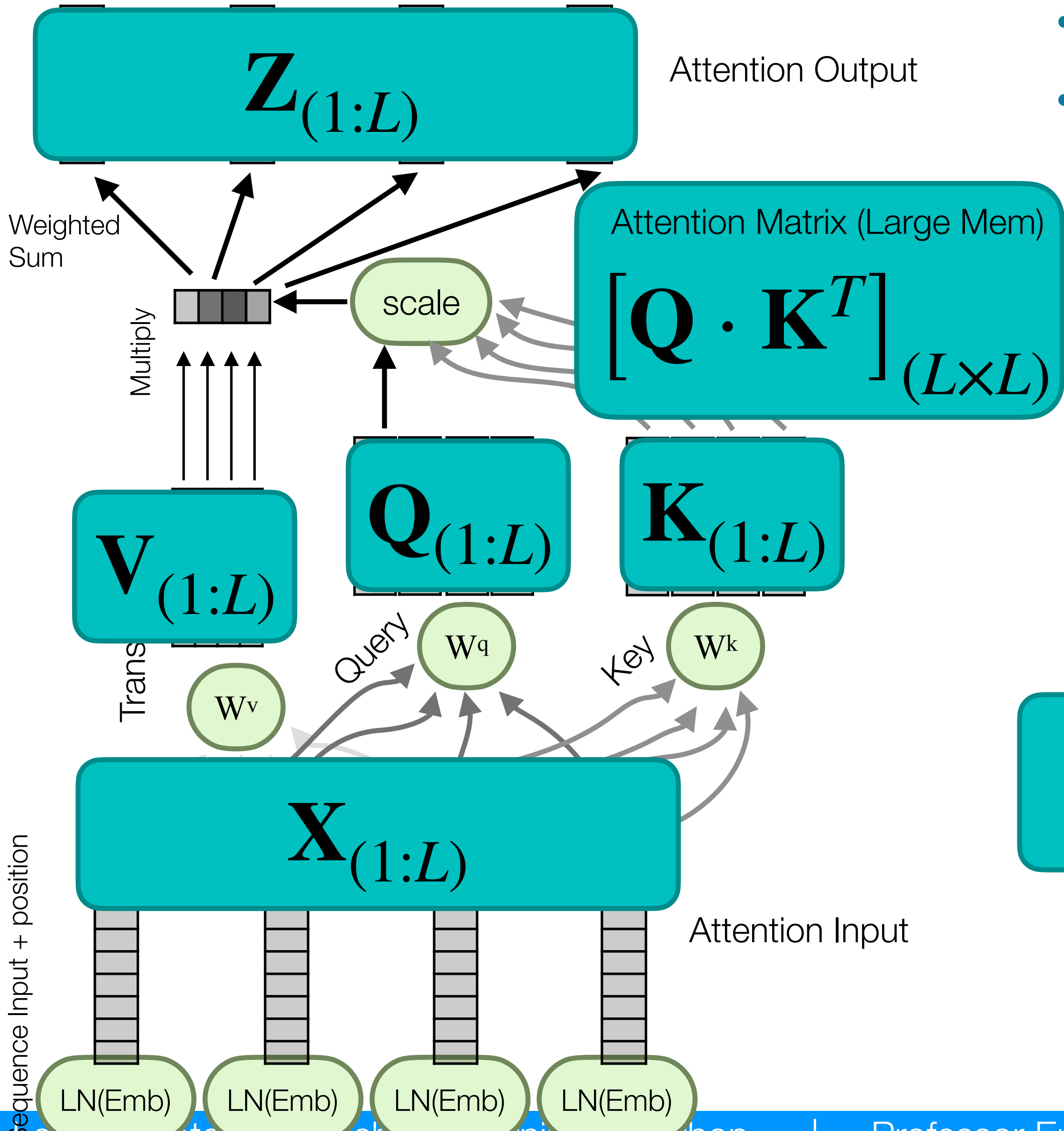
Linear and Logistic Regression
Numpy, Recreate API for Scikit-learn
Detailed mathematics for simple optimization
intuition for advanced optimization

Neural Networks and Back Prop.
Numpy
Detailed mathematics for NN operations

Wide and Deep Networks | **Convolutional Networks** | **Sequential Networks**
Keras, Tensorflow
Intuition, Detailed implement.

~~Ethics~~

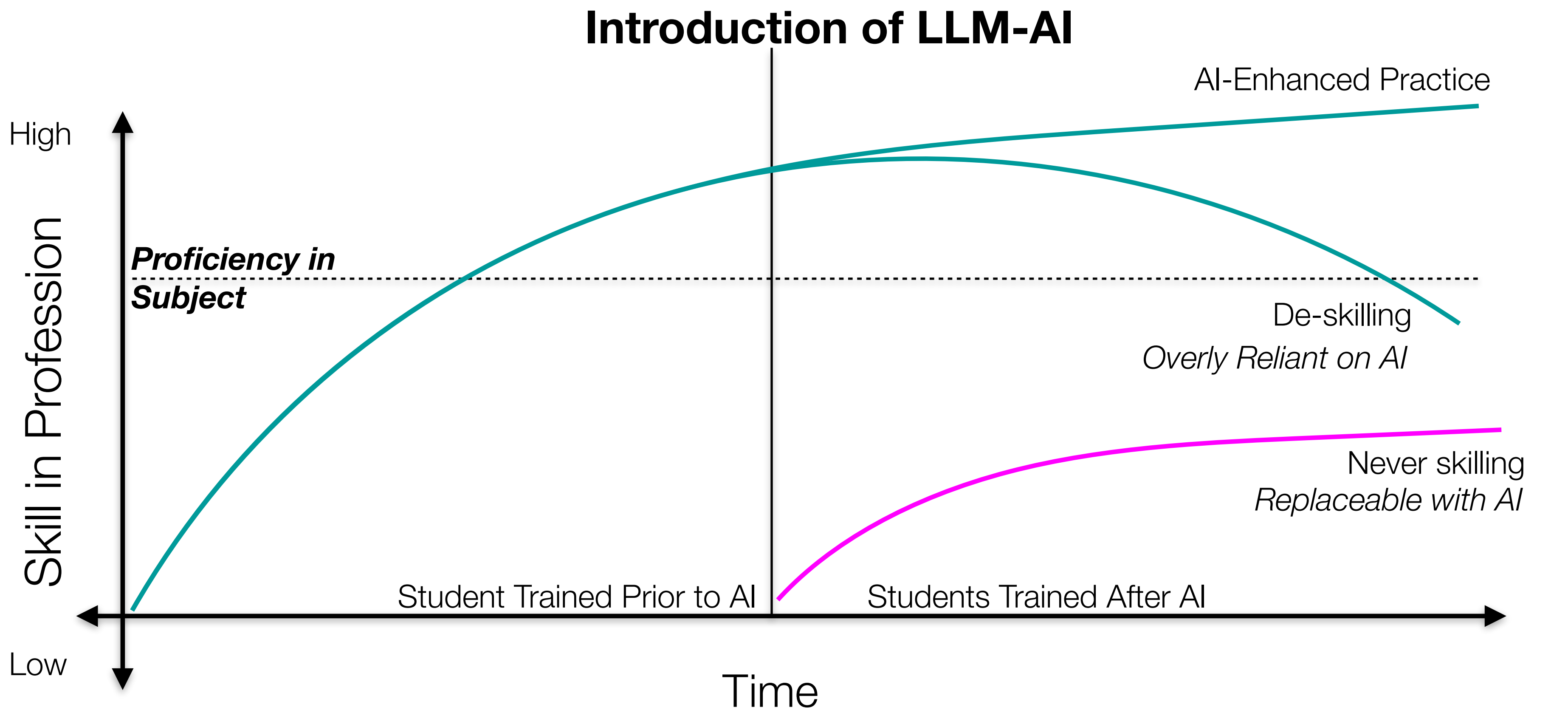
Self Attention Overview (Review)



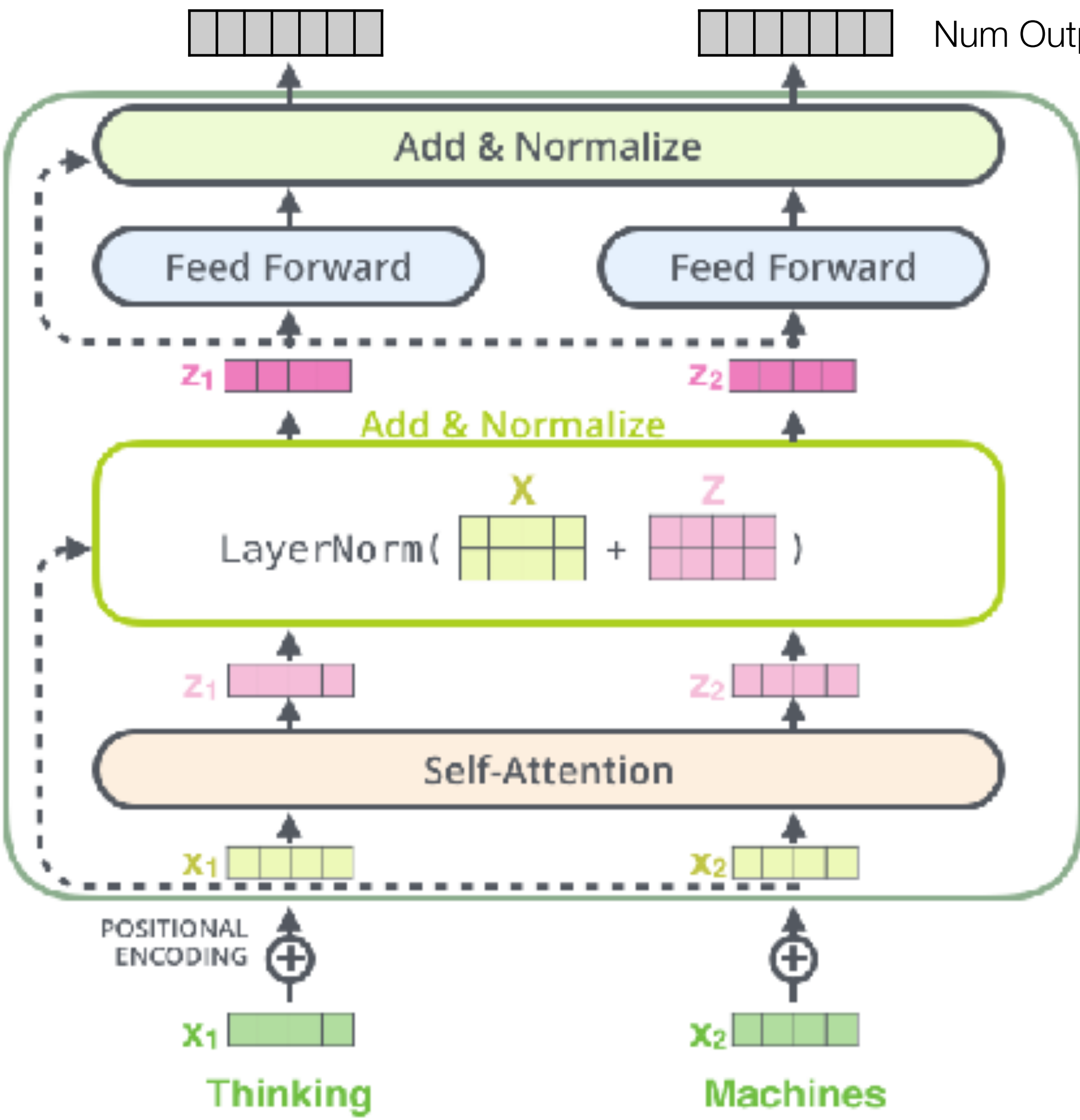
- Trained: $\mathbf{W}^v, \mathbf{W}^q, \mathbf{W}^k$
- Other Parameters:
  - L: length of sequence
  - Query/Key dimension, d_k
  - Value dimension, d_v
  - Type of positional encoding (more later)

$$\text{softmax}\left(\frac{\mathbf{Q}\cdot\mathbf{K}^T}{\sqrt{d_k}}\right)\cdot\mathbf{V}$$

Positional Encoding



Transformer for Sequence Classification



Num Outputs is Same as Inputs

Do these outputs change, if the input sequence changes order?

The order of vectors will change, but not the values of each vector…

$$\text{softmax}\left(\frac{\mathbf{Q}\cdot\mathbf{K}^T}{\sqrt{d_k}}\right)\cdot\mathbf{V}$$

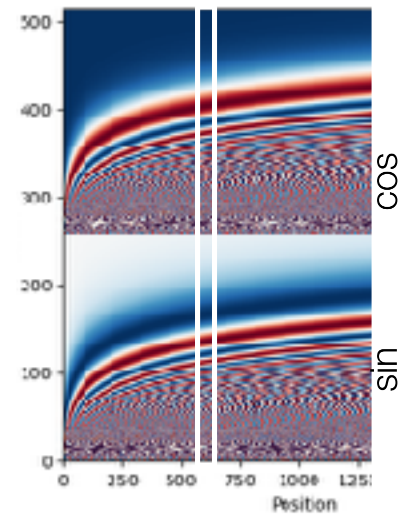
$\mathbf{V}_{(1:L)}$ $\mathbf{Q}_{(1:L)}$ $\mathbf{K}_{(1:L)}$

$\mathbf{X}_{(1:L)}$

Transformer: First Positional Encoding

- Objective: add notion of position to embedding
- Attempt in paper: add sin/cos to embedding

$$\underbrace{\hat{\mathbf{X}}}_{\text{to x-former}} = \underbrace{\mathbf{X}}_{\text{word embed}} + \underbrace{PE}_{\text{position embed}}$$

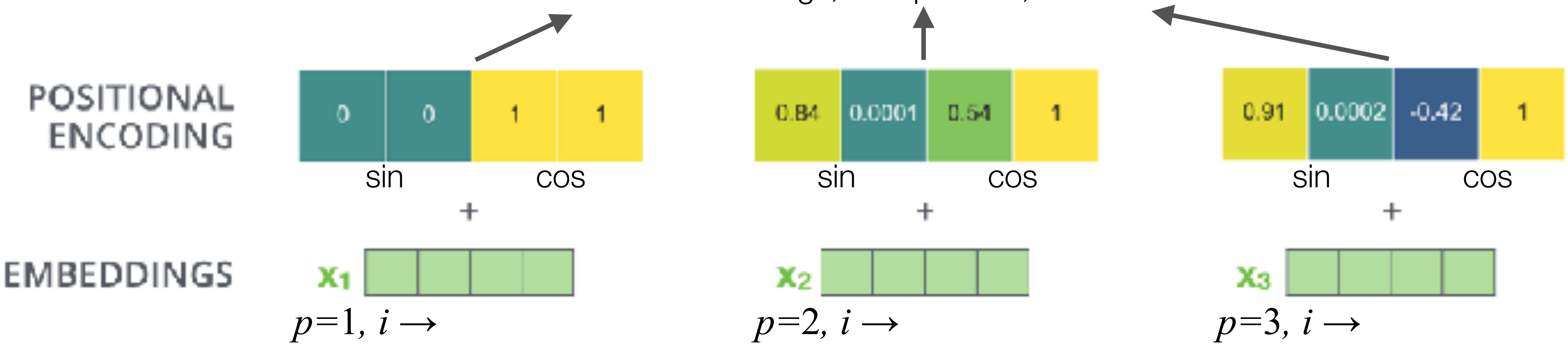


p: in sequence
D: dim of embed
i = index in vector

$$PE_{(p,i\in 0\ldots D/2-1)} = \sin(p/10000^{i/(D/2)})$$

$$PE_{(p,i\in D/2\ldots D)} = \cos(p/10000^{(i-D/2)/(D/2)})$$

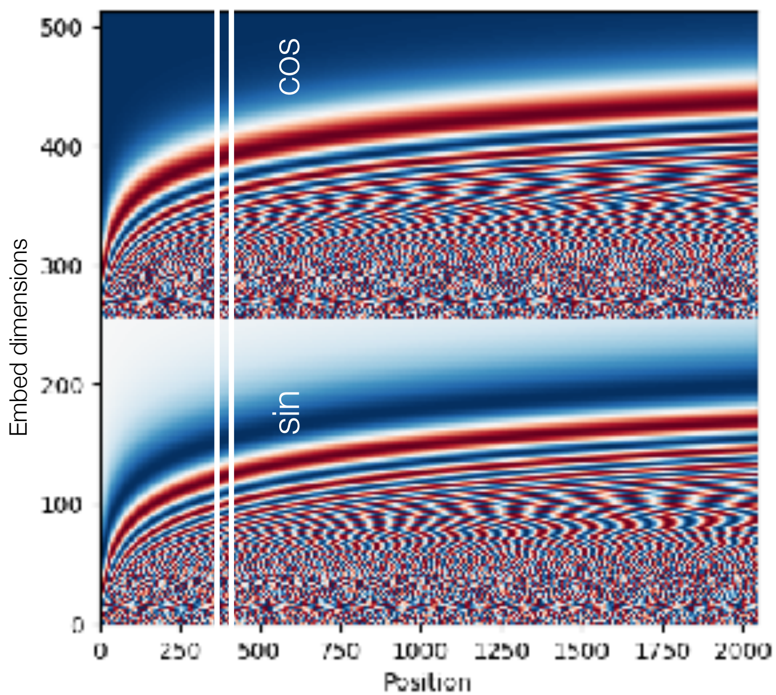
Now use the new embeddings, with position, into transformer architecture

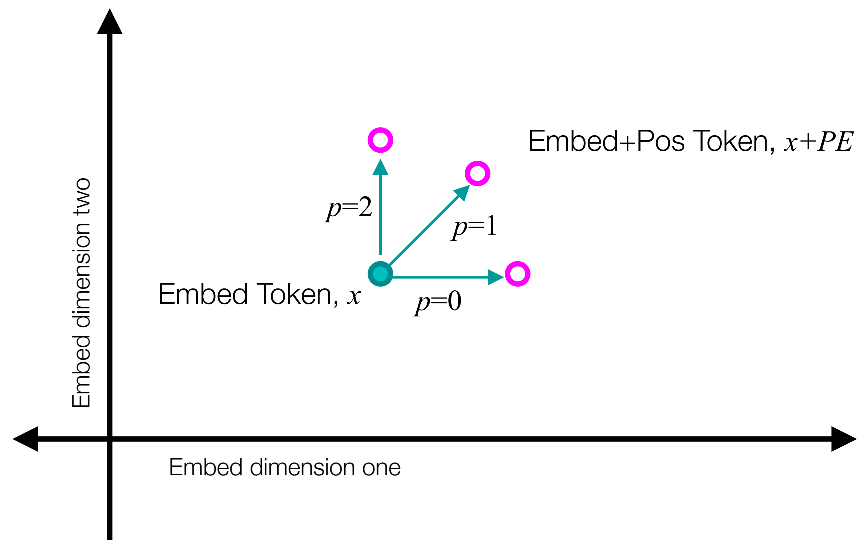


Hypothesis: Now the word proximity is encoded in the embedding matrix, with other pertinent information. Well, it does help… so it could be true that this is a good way to do it.

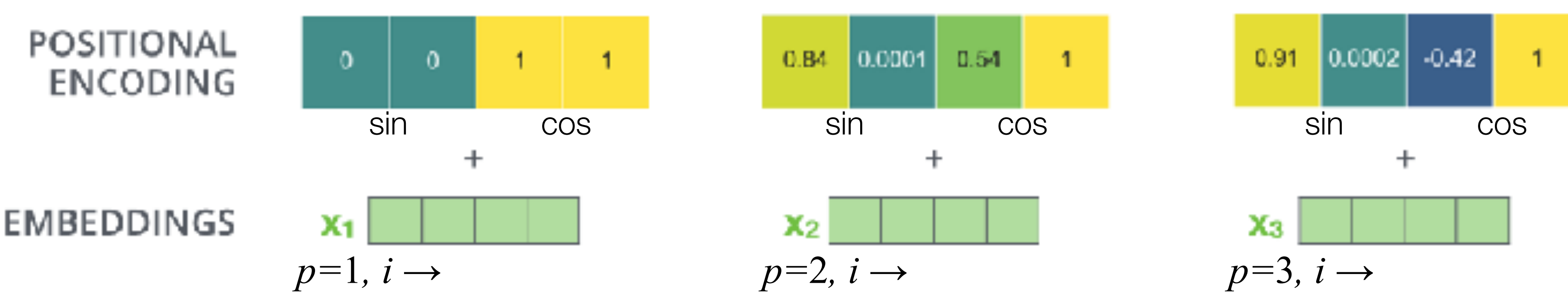
Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

Positional Intuition, Geometrically





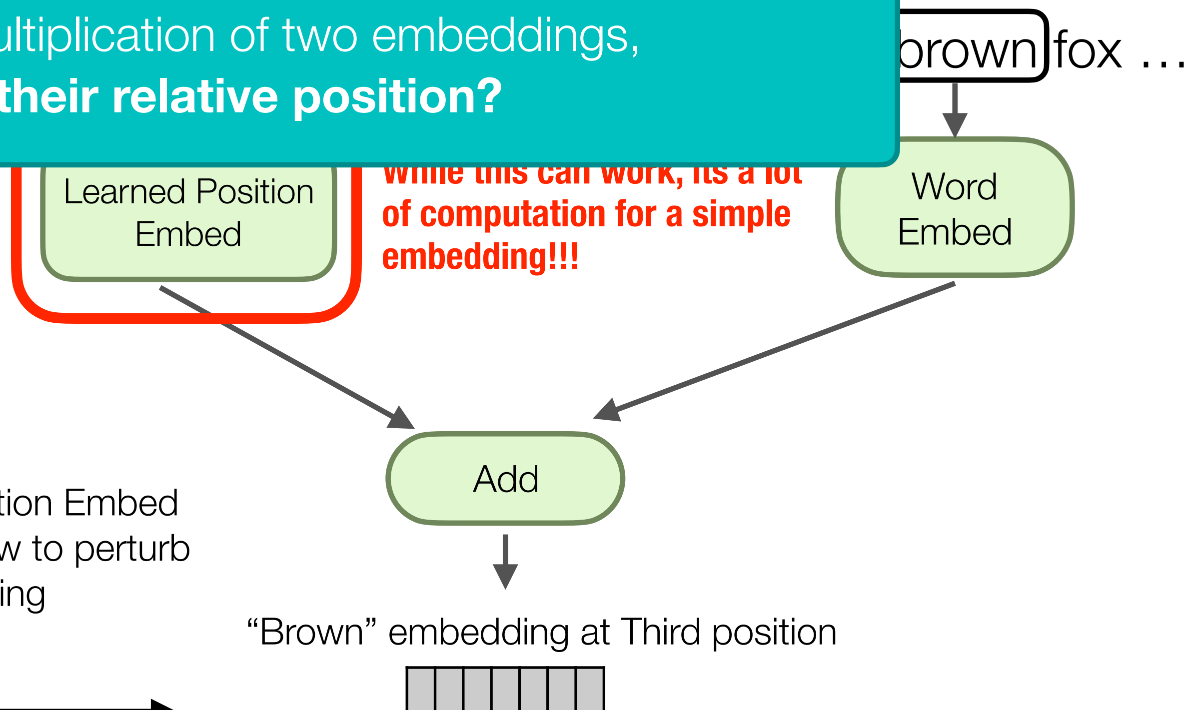
Assume that transformer model can learn the small perturbations in positional encoding, and how to use them in the context of the word.

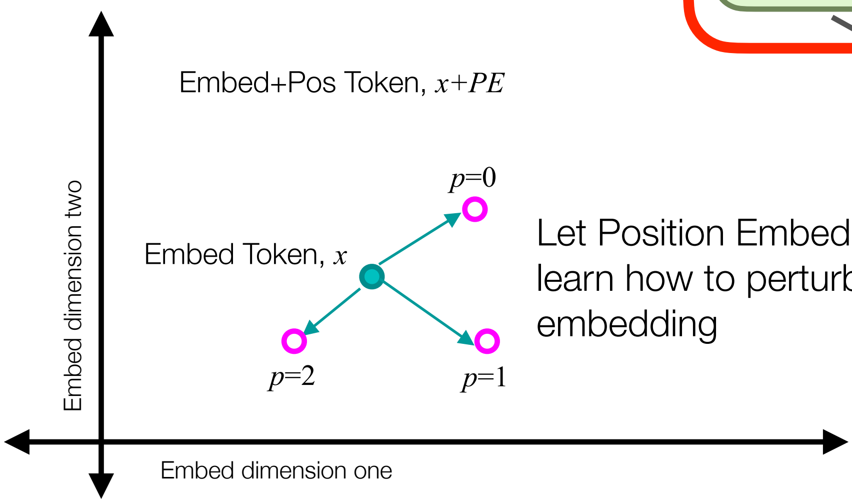


Transformer: Positional Embedding

- Objective: add notion of position to embedding
- Attempt in original paper: add sin/cos to embedding
- **But could be anything that encodes position, like:**

What if, instead of perturbing each embedding based on position,
we perturb the multiplication of two embeddings,
based on their relative position?



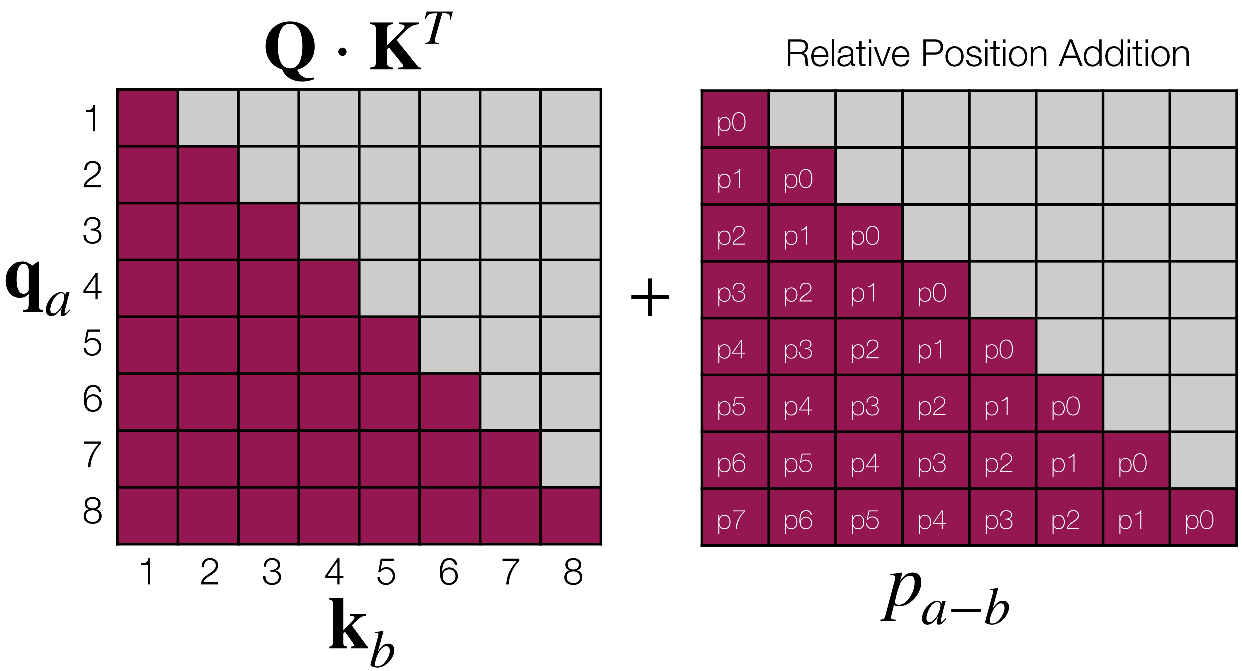


Let Position Embed learn how to perturb embedding

Excellent Blog on Transformers: http://jalammar.github.io/illustrated-transformer/

Relative Positional Encoding

- Don't encode raw position, encode relative position between $\mathbf{q}_a$ and $\mathbf{k}_b$ at position a and b
- practically, attention → softmax($\mathbf{Q} \cdot \mathbf{K}^T + p_{a-b}$)



- (+) Nicely structured relative position information
- (-) Lots more memory
- (-) Couples position and attention (harder to distribute ops)

- **How might we still encode relative position, without all the overhead?**

Smart relative position encoding

- *Ideally*, for sequence $\left[\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_a, \ldots \mathbf{x}_b, \ldots \mathbf{x}_L\right]$, relative position attention (positions a and b) is given by:

$$f(\mathbf{x}_a, \mathbf{x}_b, a-b) = \left(\underbrace{\mathbf{W}^{(q)} \cdot \mathbf{x}_a}_{\mathbf{q}_a}\right)^T R_{a-b} \left(\underbrace{\mathbf{W}^{(k)} \cdot \mathbf{x}_b}_{\mathbf{k}_b}\right)$$

$$= \mathbf{q}_a^T \cdot R_{a-b} \cdot \mathbf{k}_b$$

$$= \mathbf{q}_a^T \cdot R_a R_b \cdot \mathbf{k}_b$$

When Multiplied,
still encodes relative pos

$$= (\mathbf{q}_a^T \cdot R_a)(R_b \cdot \mathbf{k}_b) = \hat{\mathbf{q}}_a^T \cdot \hat{\mathbf{k}}_b$$

Still about as
Efficient as Attention

Precompute

$$(\mathbf{q}_a \cdot R_a) \rightarrow \hat{\mathbf{q}}_a$$
$$(R_b \cdot \mathbf{k}_b) \rightarrow \hat{\mathbf{k}}_b$$

softmax$(\hat{\mathbf{q}}_a^T \cdot \hat{\mathbf{k}}_b)$

softmax $\left(f(\mathbf{x}_a, \mathbf{x}_b, a-b)\right)$

relative position attention

- what if $\boxed{R_{a-b}}$ can be decoupled into $\boxed{R_a R_b}$ for efficiency?

$$R_a = e^{j \cdot \theta a} \qquad R_b = e^{-j \cdot \theta b} \qquad R_a R_b = e^{j \cdot \theta(a-b)} = R_{a-b}$$

one solution: rotations in the complex plain, eigenfunctions

Smart relative position encoding

- but practically $R_a R_b = e^{j \cdot \theta(a-b)} = R_{a-b}$ requires complex valued arithmetic (not ideal for efficiency). So we can get the same benefit via:

$$f(\mathbf{x}_a, \mathbf{x}_b, a-b) = \mathrm{Re}[\mathbf{q}_a^T \cdot R_a R_b \cdot \mathbf{k}_b] = \mathbf{q}_a^T \cdot \mathrm{Re}[R_a R_b] \cdot \mathbf{k}_b$$

- which in the 2 element case reduces to the rotation matrix:

$$R_a \cdot \mathbf{q}_a = \underbrace{\begin{bmatrix} \cos(a \cdot \theta) & -\sin(a \cdot \theta) \\ \sin(a \cdot \theta) & \cos(a \cdot \theta) \end{bmatrix}}_{\text{Rotate } \boldsymbol{q} \text{ by } a\,\theta\text{, precomputed}} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \qquad R_b \cdot \mathbf{k}_b = \underbrace{\begin{bmatrix} \cos(b \cdot \theta) & -\sin(b \cdot \theta) \\ \sin(b \cdot \theta) & \cos(b \cdot \theta) \end{bmatrix}}_{\text{Rotate } \boldsymbol{k} \text{ by } b\,\theta\text{, precomputed}} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}$$

- and we effectively get relative position encoding for $(\mathbf{q}_a^T \cdot R_a)(R_b \cdot \mathbf{k}_b)$, but with decoupled operations!!!
- but, expanding beyond 2D vectors starts to make the operation too computational… so let's only do operation in pairs along $\mathbf{q}$ and along $\mathbf{k}$ separately (lots of 2D rotations)

Rotary Position Encoding, RoPE

- Now we can finally understand RoPE:

$$R_a = \begin{bmatrix} \cos(a \cdot \theta_1) & -\sin(a \cdot \theta_1) & 0 & 0 & \dots & 0 & 0 \\ \sin(a \cdot \theta_1) & \cos(a \cdot \theta_1) & 0 & 0 & \dots & 0 & 0 \\ & & \cos(a \cdot \theta_2) & -\sin(a \cdot \theta_2) & \dots & & \\ & & \sin(a \cdot \theta_2) & \cos(a \cdot \theta_2) & \dots & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & & & 0 & \cos(a \cdot \theta_{D/2}) & -\sin(a \cdot \theta_{D/2}) \\ 0 & 0 & \dots & 0 & 0 & \sin(a \cdot \theta_{D/2}) & \cos(a \cdot \theta_{D/2}) \end{bmatrix}$$

Rotate by $a \cdot \theta_1$ *(large rotate)*

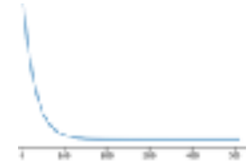
Rotate by $a \cdot \theta_2$

Rotate by $a \cdot \theta_{D/2}$ *(small rotate)*

In general, produces **better results** (mostly) while being **not too computational** (a "pretty good" relative encoding)

Lots of pairwise rotations, each preserving the property of relative position encoding

$\theta_i = 10000^{-2(i-1)/D}$ defines the range of rotations, D is size of q,k



High frequency, sensitive to position

Speed: Fast to implement with two point wise vector multiplies and addition (low overhead, still parallel)

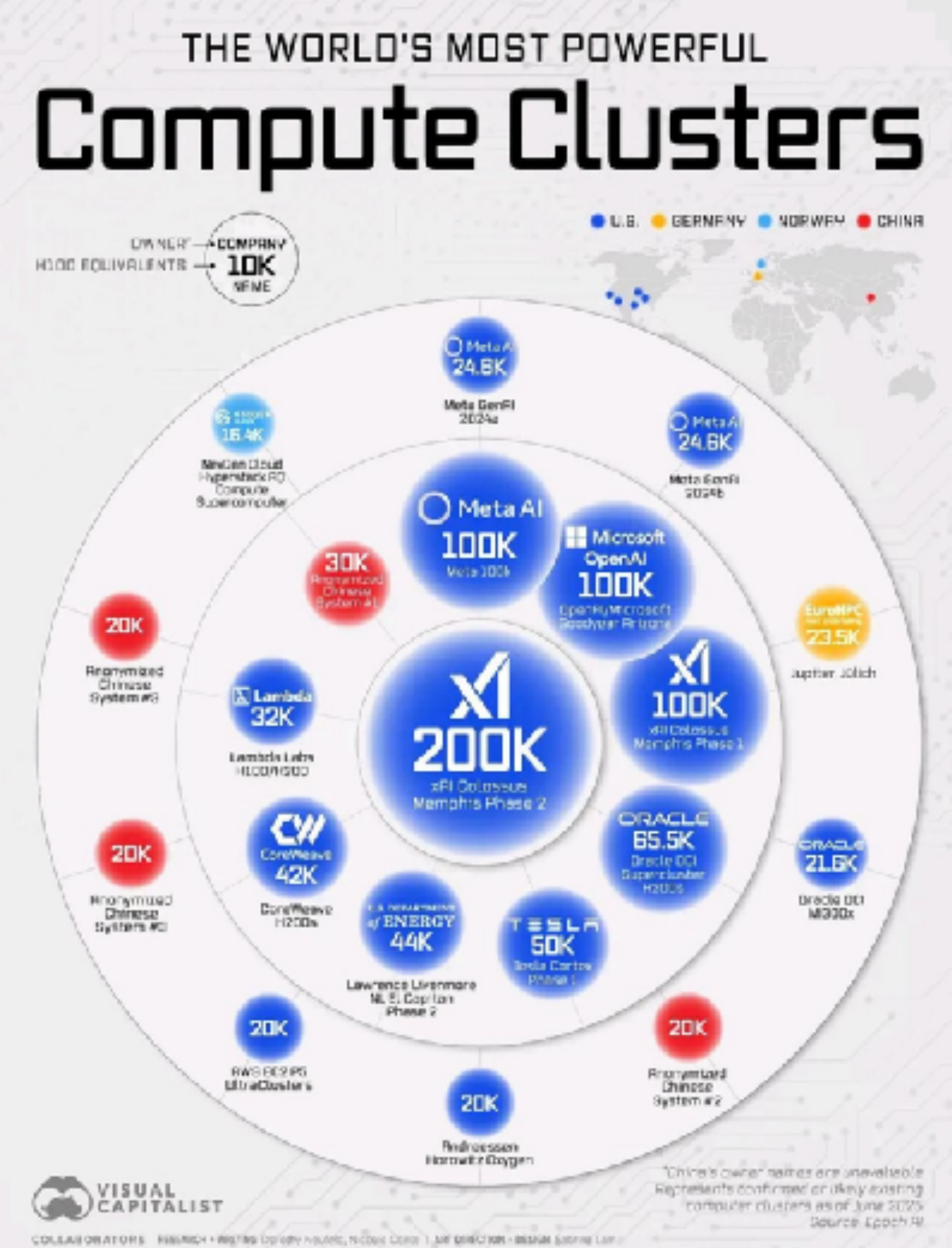
$$R_a \cdot \mathbf{q}_a = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ \vdots \\ q_{D-1} \\ q_D \end{bmatrix} \cdot \begin{bmatrix} \cos(a \cdot \theta_1) \\ \cos(a \cdot \theta_1) \\ \cos(a \cdot \theta_2) \\ \cos(a \cdot \theta_2) \\ \vdots \\ \cos(a \cdot \theta_{D/2}) \\ \cos(a \cdot \theta_{D/2}) \end{bmatrix} + \begin{bmatrix} -q_2 \\ q_1 \\ -q_4 \\ q_3 \\ \vdots \\ -q_D \\ q_{D-1} \end{bmatrix} \cdot \begin{bmatrix} \sin(a \cdot \theta_1) \\ \sin(a \cdot \theta_1) \\ \sin(a \cdot \theta_2) \\ \sin(a \cdot \theta_2) \\ \vdots \\ \sin(a \cdot \theta_{D/2}) \\ \sin(a \cdot \theta_{D/2}) \end{bmatrix}$$

Transformer learns to encode positionally sensitive aspects in high frequency indices…

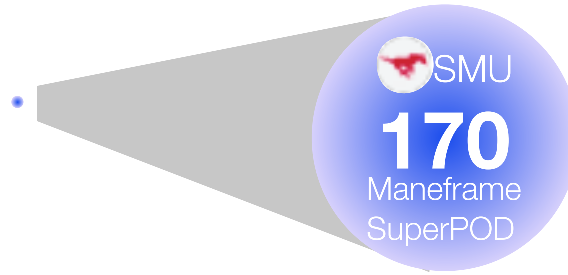
Low frequency, less sensitive to position

Su et al., RoFormer, ACM Neurocomputing (2023) https://arxiv.org/pdf/2104.09864

78



Altering Self-Attention



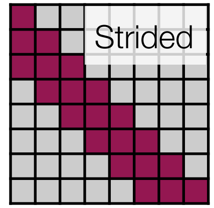
Attention Efficiently

Partial Global + Rand + Strided
Zaheer et al., **BigBird**, NeurIPS 2021

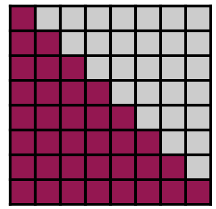
Naive Implementation:

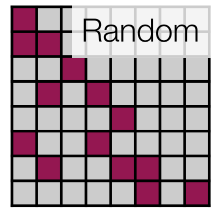
- Computation: $O(L^2 \cdot d)$
- Memory: $O(L^2 + L \cdot d)$

One idea: limit non-zero values of $\mathbf{Q} \cdot \mathbf{K}^T$
Need to define sparsity before computation



$$\text{softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^T}{\sqrt{d}}\right) \cdot \mathbf{V}$$





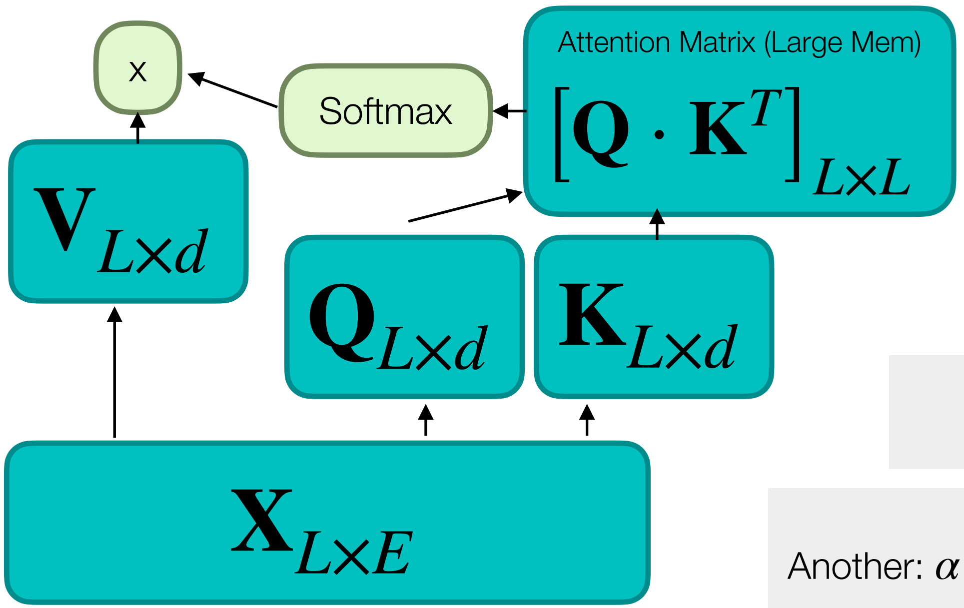
Another idea: change softmax, to allow associative rule application

$$\left(\mathbf{Q} \cdot \mathbf{K}^T\right) \cdot \mathbf{V} = \mathbf{Q} \cdot \left(\mathbf{K}^T \cdot \mathbf{V}\right)$$

$$O(L^2 \cdot d) \qquad\qquad O(L \cdot d^2)$$

Efficient Implementation:

- Computation: $O(L \cdot d^2)$
- Memory: $O(d^2 + L \cdot d)$



E: token embedding size, ***L***: sequence length,
d: transformer dimension of ***q***, ***k***, ***v***

but we need a function that satisfies

$f\left(\mathbf{Q} \cdot \mathbf{K}^T\right) = f(\mathbf{Q}) \cdot f(\mathbf{K}^T)$ *linearized attention*

One function: softmax along rows and columns

$\text{softmax}(\mathbf{Q}) \cdot \left(\text{softmax}(\mathbf{K})^T \cdot \mathbf{V}\right)$

Exponential Linear Unit
$f(x) = \max(0,(e^x - 1) \cdot \alpha)$

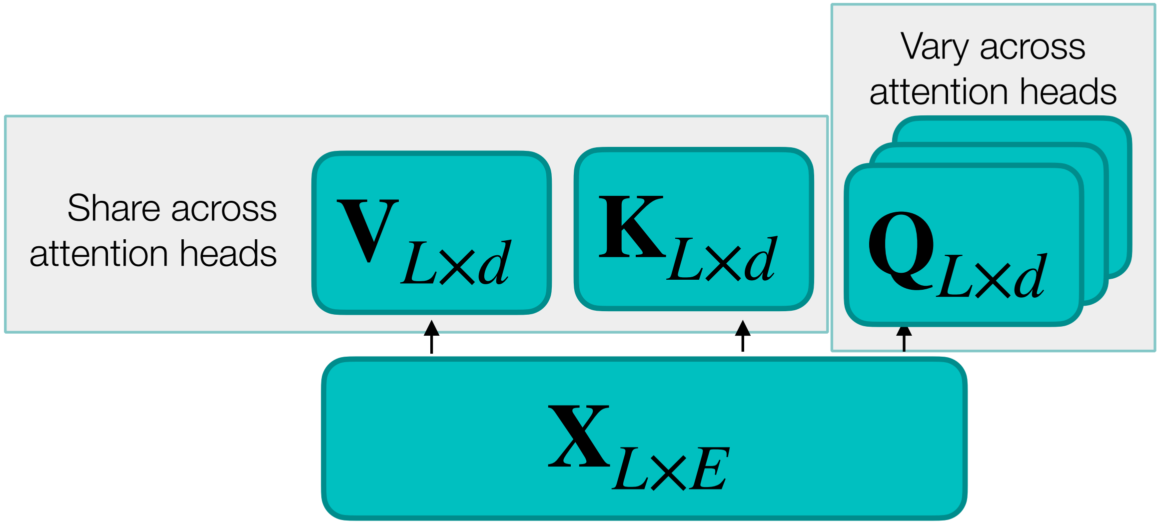
Katharopoulos et al., ICLR 2021

Another: $\alpha \dfrac{\mathbf{Q}}{||\mathbf{Q}||} \cdot \left(\dfrac{\mathbf{K}}{||\mathbf{K}||}^T \cdot \mathbf{V}\right) \rightarrow \alpha \dfrac{\mathbf{Q} \cdot \mathbf{K}^T}{||\mathbf{Q}||\,||\mathbf{K}||} \cdot \mathbf{V}$

same as cosine similarity

Mongaras, Dohm, and Larson, **Cottention**, CC 2025

Efficiency, Multi-query Attention (MQA)

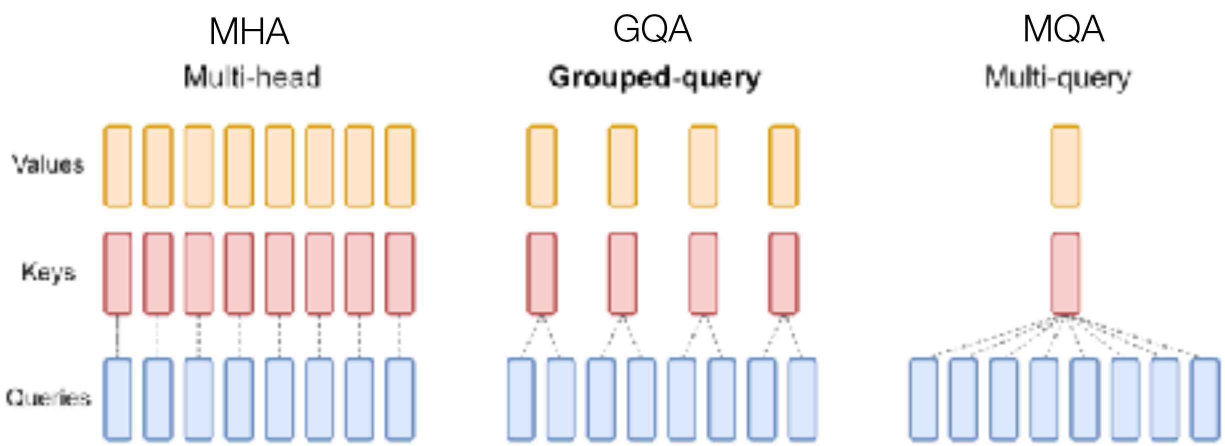


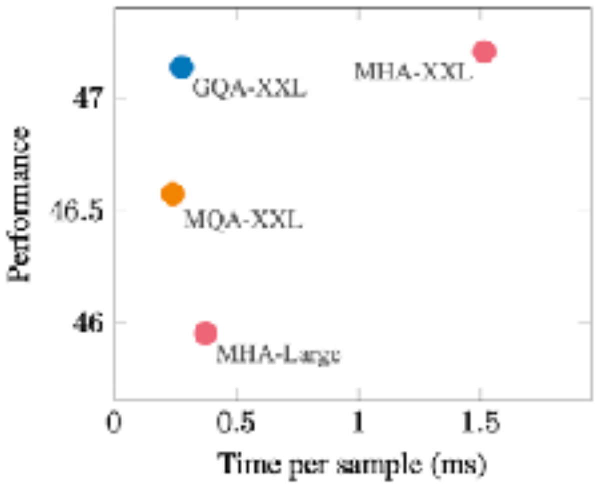
- (-) Slight drop in accuracy for various tasks
- (+) Allows larger transformer feed forward layers
- (+) larger context lengths fit in GPU memory
- (-) No speed up for distributed compute as K, V are copied

- Vanilla transformer can store V and K on SRAM of GPU, then just load in Q from high-bandwidth memory (HBM)
  - memory transfer is critical bottleneck for GPU, so you get a huge speed up
- For linear methods that can calculate $\mathbf{Q}_i \cdot \left(\mathbf{K}^T \cdot \mathbf{V}\right)$, the entire $\mathbf{K}^T \cdot \mathbf{V}$ matrix can be precomputed (*may not fit in SRAM for long sequences*)

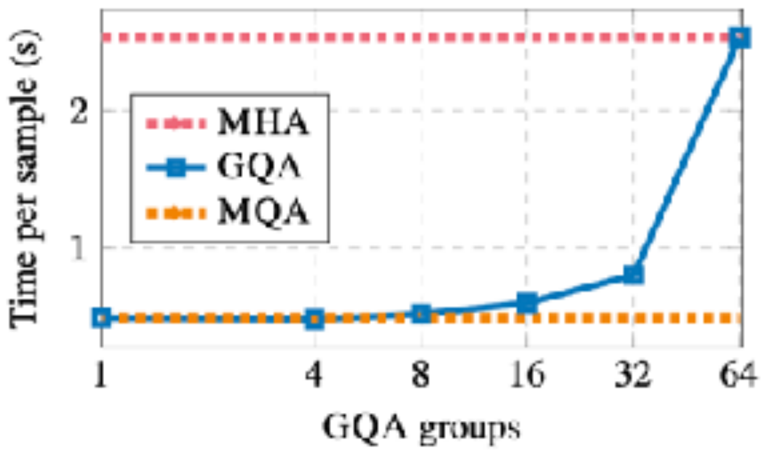
Chelba, Fast Transformer Decoding, ArXiV (2020)

Group Query Attention





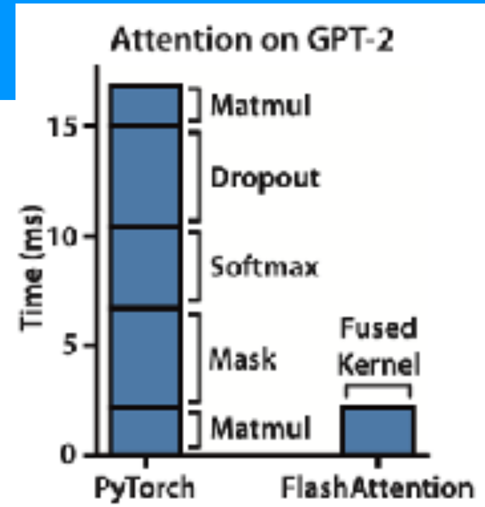
- Can take advantage of distributed computation, parallelize across groups for unique K, V
- Easy to tradeoff performance of MHA with compute of MQA

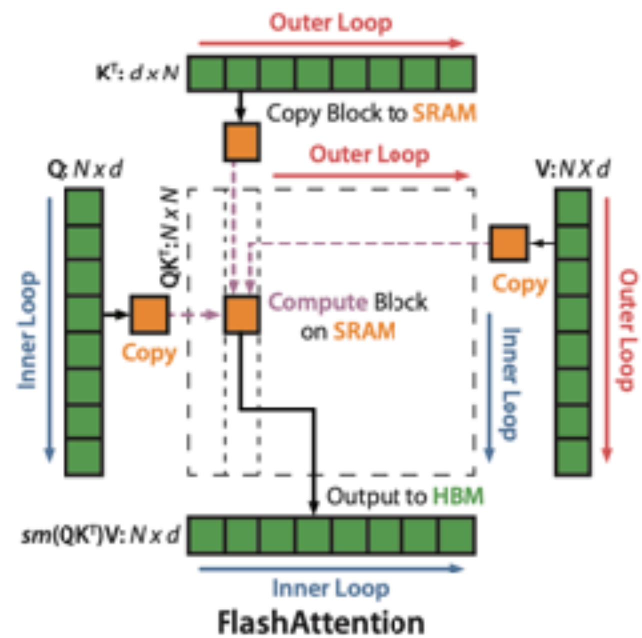


Ainslee et al., GQA, ArXiV (2023)

Flash Attention

- Calculate attention in tiles (local compute)
  - Requires calculation and saving additional variables in each tile
- During tile aggregation, scale the variables properly to get exact softmax across tiles (distributed softmax)
- Tile calculation is a shader function, massive speed up on a GPU
- Back-prop: Only save the attention output and recompute it for back propagation to save memory
  - similar to gradient checkpointing, this adds compute but saves memory
- **Flash Attention 2:**
  - Some small improvements to matrix multiplications
  - Added support for GQA (big speed ups)
  - Flash Attention becomes 9x faster than normal attention for both training and inference





Dao et al., FlashAttention, NeurIPS 2022

Distributing Softmax in Tiles

$$\mathbf{x} = [a, b, c, d]$$

$$m(\mathbf{x}) = \max\left([a, b, c, d]\right)$$

$$f(\mathbf{x}) = \left[e^{a-m(\mathbf{x})}, e^{b-m(\mathbf{x})}, e^{c-m(\mathbf{x})}, e^{d-m(\mathbf{x})}\right]$$

$$l(\mathbf{x}) = \sum f(\mathbf{x})$$

$$\text{softmax}(\mathbf{x}) = \frac{f(\mathbf{x})}{l(\mathbf{x})} = \left[\frac{e^{a-m(\mathbf{x})}}{l(\mathbf{x})}, \frac{e^{b-m(\mathbf{x})}}{l(\mathbf{x})}, \frac{e^{c-m(\mathbf{x})}}{l(\mathbf{x})}, \frac{e^{d-m(\mathbf{x})}}{l(\mathbf{x})}\right]$$

Regular Softmax Calculation

$$\mathbf{x}^{(1)} = [a, b] \quad f(\mathbf{x}^{(1)}) = \left[e^{a-m(\mathbf{x}^{(1)})}, e^{b-m(\mathbf{x}^{(1)})}\right]$$

$$\mathbf{x}^{(2)} = [c, d] \quad f(\mathbf{x}^{(2)}) = \left[e^{c-m(\mathbf{x}^{(2)})}, e^{d-m(\mathbf{x}^{(2)})}\right]$$

$$m(\mathbf{x}) = \max\left(m(\mathbf{x}^{(1)}), m(\mathbf{x}^{(2)})\right)$$

Need to track this

$$s_i = e^{m(\mathbf{x}^{(i)})-m(\mathbf{x})}$$

Slightly more FLOPS, but better utilization of parallelism

$$f(\mathbf{x}) = \left[s_1 \cdot f\left(\mathbf{x}^{(1)}\right), s_2 \cdot f\left(\mathbf{x}^{(2)}\right),\right]$$

$$l(\mathbf{x}) = s_1 \cdot l\left(\mathbf{x}^{(1)}\right) + s_2 \cdot l\left(\mathbf{x}^{(2)}\right)$$

Combine distributed ops:

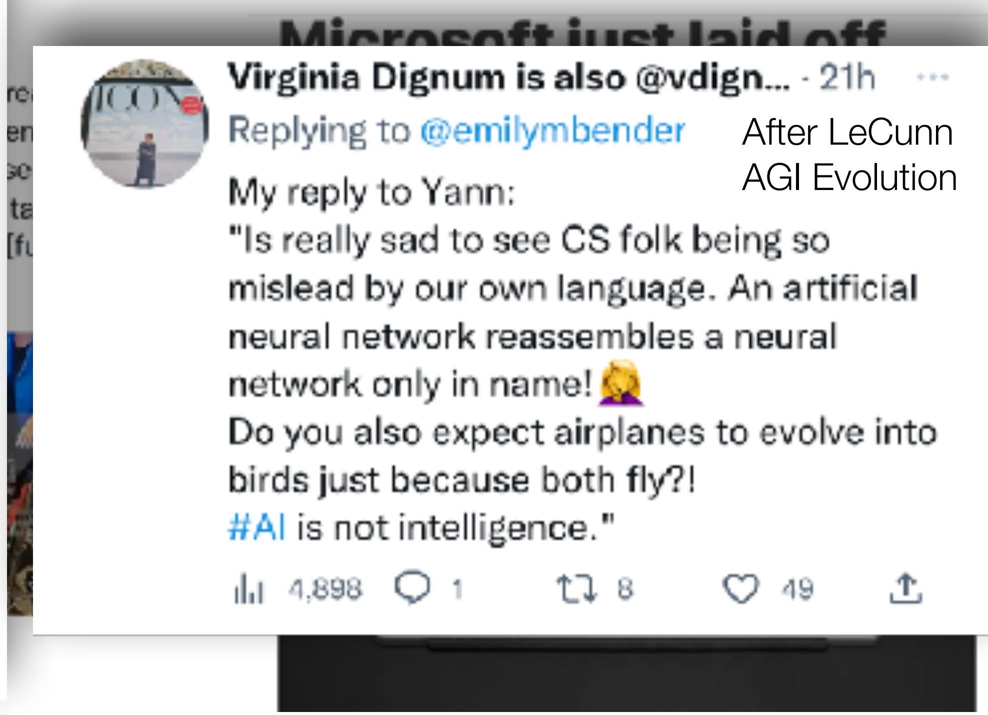
$$\text{softmax}(\mathbf{x}) = \frac{f(\mathbf{x})}{l(\mathbf{x})}$$

Distributed Softmax

End of lecture material

Course Retrospective

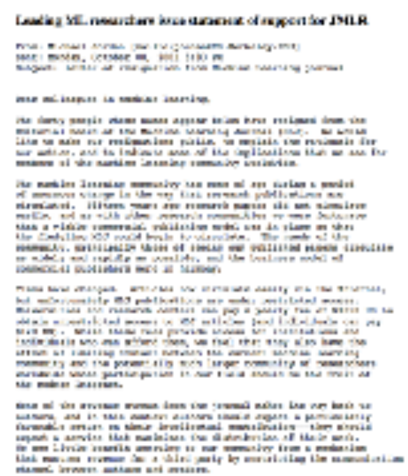
Hinton’s Student



After LeCunn
AGI Evolution

Course Retrospective

- AI winters existed because we **over-hyped** power of machine learning, imprecise wording, … for money and power, and greed
  - **and history will repeat**
- Formal methods around deep learning are **not as formal as other fields**
- At the end of the day, we are still using **back propagation**
- **Open source** is a big reason we can still have advancements without formalism:
  - http://www.jmlr.org/statement.html


Leading ML researchers issue statement of support for JMLR


Sincerely yours,

Chris Atkeson
Peter Bartlett
Andrew Barto
Jonathan Baxter
Yoshua Bengio
Kristin Bennett
Chris Bishop
Justin Boyan
Carla Brodley
Claire Cardie
William Cohen
Peter Dayan
Tom Dietterich
Jerome Friedman
Nir Friedman
Zoubin Ghahramani
David Heckerman
Geoffrey Hinton
Haym Hirsh
Tommi Jaakkola
Michael Jordan
Leslie Kaelbling
Daphne Koller
John Lafferty
Sridhar Mahadevan
Marina Meila
Andrew McCallum
Tom Mitchell
Stuart Russell
Lawrence Saul
Bernhard Schoelkopf
John Shawe-Taylor
Yoram Singer
Satinder Singh
Padhraic Smyth
Richard Sutton
Sebastian Thrun
Manfred Warmuth
Chris Williams
Robert Williamson

Topics review

Setup, Review

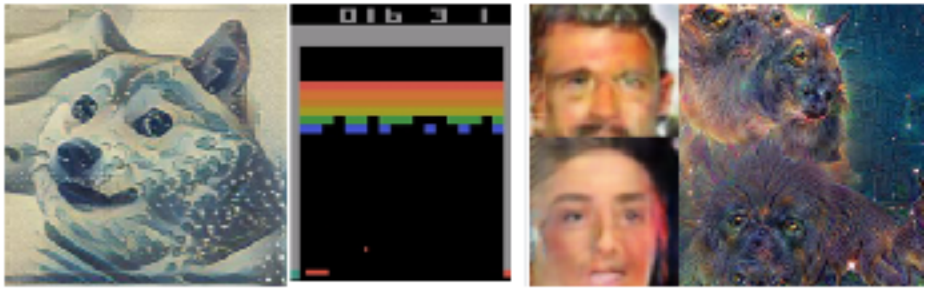
- Data **munging** in pandas and numpy and **visualization** with matplotlib, pandas, seaborn
- Data preprocessing: **dim reduction**, images, text, categorical features, **embeddings**
- **Linear models**: linear regression, logistic regression, simple neural networks

Deep Learning

- **Optimization** strategies: Gradient ascent, Quasi-Newton, Extensions of SGD (RMSProp, AdaM)
- **Back propagation** in MLP (from scratch)
- Tensorflow/Keras for **wide and deep networks**
- **Convolutional** neural networks (up to modern day)
- **Sequential** neural networks (transformers, CNNs)

Topics Not Covered

- Methods to Assist Ethics in ML
- Transfer/Multi-Task Learning
- Visualizing CNNs
- Fully Convolutional Networks
- Style Transfer (maybe)
- Generative Networks
- Large Language Models



Learning and Neural Networks

Overview

This course extends basic knowledge of the use of Neural Networks in machine learning beyonds simple prediction, especially targeted outputs that are generation or alteration of images, text, and audio. This course emphasizes topics of neural networks in the "deep learning" subdomain. This course will survey of important topics and current areas of research, including transfer learning, multi-task and multi modal learning, image style transfer, neural network visualization, deep convolutional generative adversarial networks, and deep reinforcement learning. For grading, students are expected to complete smaller team-based projects throughout the semester, present one research paper in a 15-20 minute group presentation (covering topics in the course), and complete a comprehensive final project that involves a number of different deep learning architectures.

Thank you for a great semester!

- but it could **have been better** somehow, right?
  - How could you learn better, more reliably for an interview?
  - When did you feel like you **built proficiency**? Versus when was **critical thought** not attained?
  - what should **not be cut** or **not changed**?
    - **Already cut**: SVMs, Random forests, Boosting, Ensembles, RNNs, many-to-many,
  - Two courses: (1) Intro ML and (2) Deep Learning
  - More APIs? Turi / PyTorch?
  - More flipped Assignments?
  - Fewer coding demos?
    - Self-guided Jupyter notebooks?
  - Exams?

Thank You for an Excellent Semester!

Courtesy of Omar Roa

Please fill out the course evaluations!!!!