Large Language Models

Narayana Santhanam

EE 645
Mar 24, 2025

University of Hawai'i Mānoa

This module

- Latent Semantic Indexing
  - SVD
- Language models (Transformers)
  - In context learning
  - Non-language data

Latent Semantic Indexing

Singular value decomposition

$$M = U\Sigma V^T$$

If M is $n \times p$,

U is $n \times n$

Σ is $n \times p$

V is $p \times p$

Singular value decomposition

$$M = U\Sigma V^T$$

If M is $n \times p$,

U is $n \times n$

Σ is $n \times p$

V is $p \times p$

U, V are both orthonormal

$U^T = U^{-1}$ and $V^T = V^{-1}$

Singular value decomposition

$$M = U\Sigma V^T$$

If M is $n \times p$,

U is $n \times n$

Σ is $n \times p$

V is $p \times p$

U, V are both orthonormal

$U^T = U^{-1}$ and $V^T = V^{-1}$

Σ is diagonoal

all diagonal entries ≥ 0

(called singular values)

Singular value decomposition

M is $n \times p$,

$$M = U\Sigma V^T$$

cols of U: basis for cols of M

$U = \begin{bmatrix} \mathrm{u}_1 & \cdots & \mathrm{u}_n \end{bmatrix}$, each $\mathrm{u}_i \in \mathbb{R}^n$

u_i all have length 1, mutually perpendicular

cols of V: basis for rows of M

$V = \begin{bmatrix} \mathrm{v}_1 & \cdots & \mathrm{v}_p \end{bmatrix}$, each $\mathrm{v}_i \in \mathbb{R}^n$

v_i all have length 1, mutually perpendicular

singular values: importance of basis vectors

$\sigma_1, \ldots, \sigma_{\min(n,p)}$

Multiplying out

M is $n \times p$,

$$M = \begin{bmatrix} u_1 & \dots & u_n \end{bmatrix} \mathrm{diag}\left(\sigma_1, \dots, \sigma_{\min(n,p)}\right) \begin{bmatrix} v_1^T \\ \vdots \\ v_p^T \end{bmatrix}$$

Instructive to multiply out:

$$M = \sigma_1 u_1 v_1^T + \dots + \sigma_{\min(n,p)} u_{\min(n,p)} v_{min(n,p)}^T$$

Multiplying out

M is $n \times p$,

$$M = \begin{bmatrix} \mathrm{u}_1 & \dots & \mathrm{u}_n \end{bmatrix} \mathrm{diag}\left(\sigma_1, \dots, \sigma_{\min(n,p)}\right) \begin{bmatrix} \mathrm{v}_1^T \\ \vdots \\ \mathrm{v}_p^T \end{bmatrix}$$

Instructive to multiply out:

$$M = \sigma_1 \mathrm{u}_1 \mathrm{v}_1^T + \ldots + \sigma_{\min(n,p)} \mathrm{u}_{\min(n,p)} \mathrm{v}_{min(n,p)}^T$$

Each of $\mathrm{u}_i \mathrm{v}_i^T$ is a rank-1 matrix

Multiplying out

M is $n \times p$,

$$M = \begin{bmatrix} \mathsf{u}_1 & \dots & \mathsf{u}_n \end{bmatrix} \operatorname{diag}\left(\sigma_1, \dots, \sigma_{\min(n,p)}\right) \begin{bmatrix} \mathsf{v}_1^T \\ \vdots \\ \mathsf{v}_p^T \end{bmatrix}$$

Instructive to multiply out:

$$M = \sigma_1 \mathsf{u}_1 \mathsf{v}_1^T + \dots + \sigma_{\min(n,p)} \mathsf{u}_{\min(n,p)} \mathsf{v}_{min(n,p)}^T$$

Each of $\mathsf{u}_i \mathsf{v}_i^T$ is a rank-1 matrix
Number of non-zero singular values = rank of matrix

Multiplying out

M is $n \times p$,

$$M = \begin{bmatrix} u_1 & \dots & u_n \end{bmatrix} \operatorname{diag}\left(\sigma_1, \dots, \sigma_{\min(n,p)}\right) \begin{bmatrix} v_1^T \\ \vdots \\ v_p^T \end{bmatrix}$$

Instructive to multiply out:

$$M = \sigma_1 u_1 v_1^T + \dots + \sigma_{\min(n,p)} u_{\min(n,p)} v_{min(n,p)}^T$$

Each of $u_i v_i^T$ is a rank-1 matrix
Number of non-zero singular values = rank of matrix
In fact, general definition of rank:

Rank of a matrix

M is defined rank-r if it can be written as a sum of r rank-1 matrices and no fewer.

Latent Semantic Indexing

p documents, total of n words in the documents

M is the $n \times p$ term-document matrix

Different ways to come up with M

simplest $M_{ij} = 1$ if word $i \in$ doc j

Note: M loses information about relative ordering of words

bag of words model

formally equivalent to unigram language models

Latent Semantic Indexing

Singular value decomposition of M (assume $\sigma_1 \geq \sigma_2 \geq ...$)

$$\begin{aligned} M &= \sigma_1 \mathrm{u}_1 \mathrm{v}_1^T + \ldots + \sigma_{\min(n,p)} \mathrm{u}_{\min(n,p)} \mathrm{v}_{min(n,p)}^T \\ &\approx \sigma_1 \mathrm{u}_1 \mathrm{v}_1^T + \ldots + \sigma_r \mathrm{u}_r \mathrm{v}_r^T \qquad (r \ll \min(n,p)) \quad = U^{(r)} {V^{(r)}}^T \end{aligned}$$

where $U^{(r)}$ ($V^{(r)}$) contains first r cols of U (V)

Latent Semantic Indexing

Singular value decomposition of M (assume $\sigma_1 \geq \sigma_2 \geq ...$)

$$
\begin{aligned}
M &= \sigma_1 \mathrm{u}_1 \mathrm{v}_1^T + \ldots + \sigma_{\min(n,p)} \mathrm{u}_{\min(n,p)} \mathrm{v}_{min(n,p)}^T \\
&\approx \sigma_1 \mathrm{u}_1 \mathrm{v}_1^T + \ldots + \sigma_r \mathrm{u}_r \mathrm{v}_r^T \qquad (r \ll \min(n,p)) \quad = U^{(r)} {V^{(r)}}^T
\end{aligned}
$$

where $U^{(r)}$ ($V^{(r)}$) contains first r cols of U (V) Interpret the r vectors $\mathrm{u}_1, \ldots, \mathrm{u}_r$ as the r topics

Latent Semantic Indexing

Singular value decomposition of M (assume $\sigma_1 \geq \sigma_2 \geq$ ...)

$$M = \sigma_1 \mathrm{u}_1 \mathrm{v}_1^T + \ldots + \sigma_{\min(n,p)} \mathrm{u}_{\min(n,p)} \mathrm{v}_{min(n,p)}^T$$

$$\approx \sigma_1 \mathrm{u}_1 \mathrm{v}_1^T + \ldots + \sigma_r \mathrm{u}_r \mathrm{v}_r^T \qquad (r \ll \min(n,p)) \quad = U^{(r)} {V^{(r)}}^T$$

where $U^{(r)}$ ($V^{(r)}$) contains first r cols of U (V) Interpret the r vectors $\mathrm{u}_1, \ldots, \mathrm{u}_r$ as the r topics
Interpret the r vectors $\mathrm{v}_1, \ldots, \mathrm{v}_r$ as choice of topics in each doc

Demo

Pros and cons

Pros

Simple and fast

Often used to optimize search

Pros and cons

Pros

Simple and fast

Often used to optimize search

Cons

Topics orthogonal?

Negative values

- signal words absent (ok!)
- docs similar using *absence* of words, (not ok!)

Non negative matrix factorization

LSI: $M \approx U^{(r)} {V^{(r)}}^T$
How about find best A, W such that

$$M \approx AW,$$

A has r cols, W has r rows, all entries ≥ 0
Lot harder than SVD, optimization NP-hard
Approximations exist (EM, algebraic)

Why did we study SVD so carefully?

While SVD is important in its own right, we have an ulterior motive

Mechanistic interpretations: the SVD of weight matrices in transformers seem to be quite interpretable

Very important for interpretability

Why did we study SVD so carefully?

While SVD is important in its own right, we have an ulterior motive

Mechanistic interpretations: the SVD of weight matrices in transformers seem to be quite interpretable

Very important for interpretability

Topic you can explore in a project

Large Language Models

Statistical models of language

Unigram, Bigram, Trigram...

Little bit of information theory (offline)

- entropy
- representation in bits
- cross entropy

Perplexity (power of a language model)

- GPT-4 2.6
- GPT-3.5 4.5

Modern Language Models

Tokenizer (▸ OpenAI)

Modern Language Models

Tokenizer (OpenAI)

Brief history:

- Recurrent NN
- LSTMs
- Transformers

Modern Language Models

Tokenizer (▸ OpenAI)

Brief history:

- Recurrent NN
- LSTMs
- Transformers
  - only focus on this!

Deep NN terminology

Layers

Batches, epochs

Optimizers are variants of gradient descent (but clever)
- ADAM and ADAMW most commonly used
- Adagrad, rmsprop

Transformers

What is a transformer?

Central to Transformers is the notion of *attention*

Attention-like approaches in

- Linear Regression
- Kernels

University of Hawaiʻi Mānoa

Transformer core

Attention

Skip connections

Layer Normalization

Embeddings

Attention-like approaches

$n \times p$ design matrix X, target y
Each row is an example (key)
Each target is a number (value)
Given a test example z (query), output?
Recall

$$\hat{w} = (X^T X)^{-1} X^T \mathrm{y}, \qquad \text{Prediction: } \mathrm{z}^T \hat{\mathrm{w}}$$

If $\mathrm{x}_1, \ldots, \mathrm{x}_n$ are the n examples:

$$\mathrm{z}^T \hat{\mathrm{w}} = \sum_{i=1}^{n} (\mathrm{z}^T (X^T X)^{-1} \mathrm{x}_i) y_i$$

Attention

The term $\mathrm{z}^T (X^T X)^{-1} \mathrm{x}_i$ is the attention the key x_i gets from the query z. The output is a linear combination of values y_i, with y_i weighted by the attention placed x_i.

Other algorithms

Ridge Regression

$$z^T \hat{w} = \sum_{i=1}^{n} (z^T (X^T X + \lambda I)^{-1} x_i) y_i$$

Other algorithms

Ridge Regression

$$z^T \hat{w} = \sum_{i=1}^{n} (z^T (X^T X + \lambda I)^{-1} x_i) y_i$$

Support vector machines

Representer Theorem $w = \sum_{i=1}^{n} \beta_i x_i y_i$ (linear)

Soft prediction

$$z^T \hat{w} = \sum_{i=1}^{n} \beta_i (z^T x_i) y_i$$

β_i is obtained by solving the dual, most are 0

Other algorithms

Ridge Regression

$$z^T\hat{w} = \sum_{i=1}^{n}(z^T(X^TX+\lambda I)^{-1}x_i)y_i$$

Support vector machines

Representer Theorem $w = \sum_{i=1}^{n}\beta_i x_i y_i$ (linear)

Soft prediction

$$z^T\hat{w} = \sum_{i=1}^{n}\beta_i(z^T x_i)y_i$$

β_i is obtained by solving the dual, most are 0

Attention

The term $\beta_i z^T x_i$ is the attention the key x_i gets from the query z. The output is a linear combination of values y_i, with y_i weighted by the attention placed x_i.

Attention in LLMs

We specialize the observation in prior slides

Attention in Deep Learning: probability distribution over keys

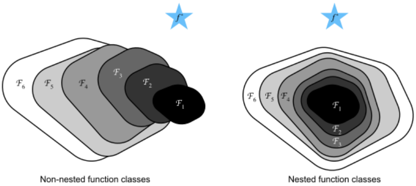
on any key must be ≥ 0

must sum to 1 over all the keys

in that sense, diff from OLS and kernel illustrations

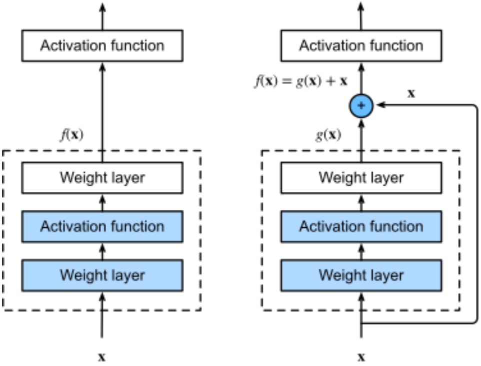
Arbitrary function and pass it through softmax

Skip connections



(Image source: Dive into deep learning)

Skip connections



(Image source: Dive into deep learning)

Layer Normalization

In deep networks, think of the input to different layers

Layer Normalization

In deep networks, think of the input to different layers

As we train, earlier layers change, new statistics in input to deeper layers

Layer Normalization

In deep networks, think of the input to different layers

As we train, earlier layers change, new statistics in input to deeper layers

different magnitudes of gradients too

Layer Normalization

In deep networks, think of the input to different layers

As we train, earlier layers change, new statistics in input to deeper layers

different magnitudes of gradients too

To address this "covariate shift", multiple normalization techniques

Batch Normalization (normalize input to layer across batches)

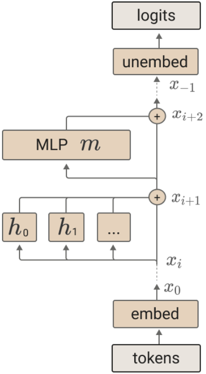
Layer Normalization (normalize input to layer across neurons)

Layer and Batch normalizations

p features, batches of size b, layer with h neurons
$Y = XW$, weight W is $p \times h$, data X is $b \times p$

	Rescale W	Center W	Rescale X	Center X
Batch	Invariant	Not inv	Invariant	Invariant
Layer	Invariant	Invariant	Invariant	Not inv

Putting them together



The final logits are produced by applying the unembedding.

$$T(t) = W_U x_{-1}$$

An MLP layer, m, is run and added to the residual stream.

$$x_{i+2} = x_{i+1} + m(x_{i+1})$$

Each attention head, h, is run and added to the residual stream.

$$x_{i+1} = x_i + \sum_{h \in H_i} h(x_i)$$

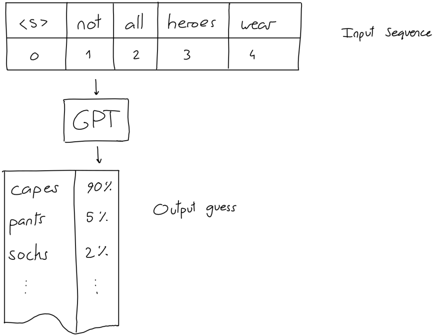
One residual block

Token embedding.

$$x_0 = W_E t$$

(Image source: A mathematical framework for transformer circuits, Anthropic)

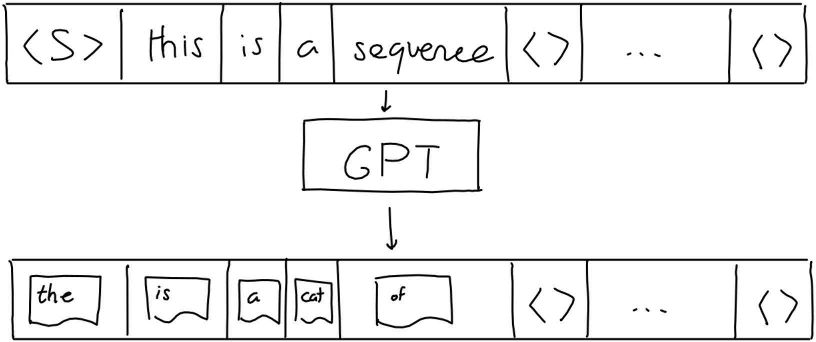
What is a Language Model?



(Image source: GPT architecture on a napkin)

What does a Transformer output?

Context has 2048 tokens (though pic shows words)



(Image source: GPT architecture on a napkin)

Representation of tokens

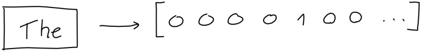
GPT has a vocabulary of 50,257 tokens

$$\boxed{\text{The}} \longrightarrow [0\ 0\ 0\ 0\ 1\ 0\ 0\ \dots]$$

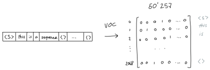
For every token in context

Representation of tokens

GPT has a vocabulary of 50,257 tokens

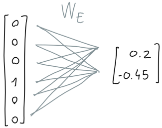


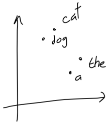
For every token in context



(Image source: GPT architecture on a napkin)

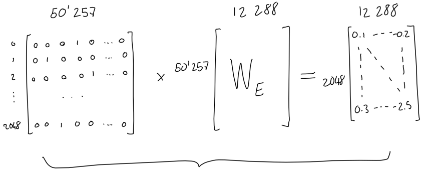
Embeding tokens

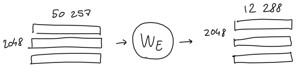




Linear map

In actuality, each token $\to \mathbb{R}^{12288}$





Positional Encoding

Each position (0-2047) $\rightarrow \mathbb{R}^{12288}$
P: position matrix (2048$\times$ 12288)

$$p_{i,2j} = \sin\left(\frac{i}{M^{2j/d}}\right)$$
$$p_{i,2j+1} = \cos\left(\frac{i}{M^{2j/d}}\right)$$

M is a large number (not important)

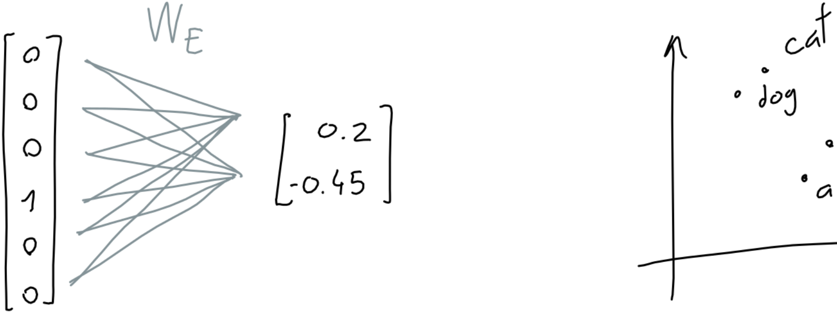
Positional Encoding

Each position (0-2047) $\to \mathbb{R}^{12288}$
P: position matrix (2048$\times$ 12288)

$$p_{i,2j} = \sin\left(\frac{i}{M^{2j/d}}\right)$$
$$p_{i,2j+1} = \cos\left(\frac{i}{M^{2j/d}}\right)$$

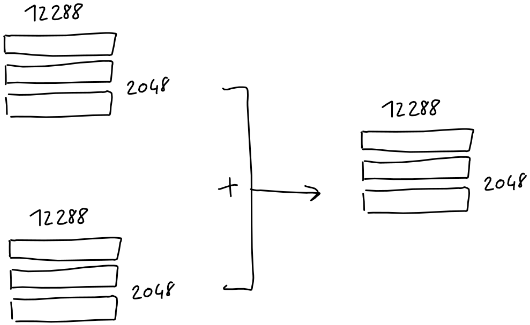
M is a large number (not important)
Idea: mimic binary representation of numbers
relative location is a linear transform

Positional encoding matrix P



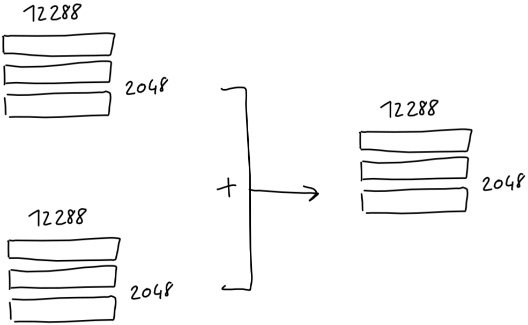
(Image source: Dive into Deep Learning)

Embedding all 2048 tokens



(Image source: GPT architecture on a napkin)

Embedding all 2048 tokens

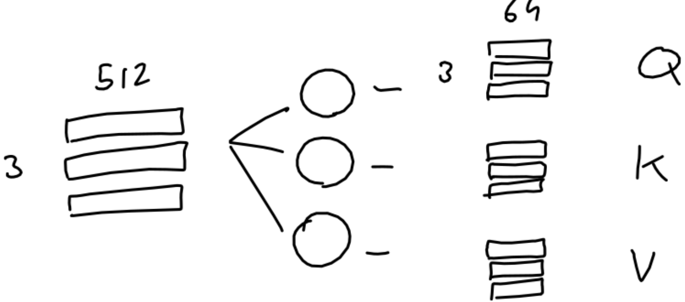


(Image source: GPT architecture on a napkin)

Embedded tokens are then split into multiple heads in each layer

GPT-3 has 96 heads

Transformer core: attention



In GPT-3: query, key, values in each head are 12288/96=128-long vectors

(Image source: GPT architecture on a napkin)

Transformer core: attention

Compute softmax$((QK^T)V)$
For query q_i from token i, compute

$$\sum_{j=1}^{n} \alpha(\mathrm{q}_i, \mathrm{k_j})\mathrm{v_j}$$

for every key k_j and value v_j from token j,

Transformer core: attention

Compute softmax$((QK^T)V)$
For query q_i from token i, compute

$$\sum_{j=1}^{n} \alpha(\mathrm{q}_i, \mathrm{k_j})\mathrm{v_j}$$

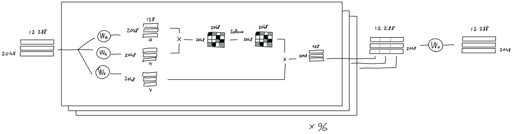
for every key k_j and value v_j from token j,

$$\alpha(\mathrm{q}_i, \mathrm{k_j}) = \mathrm{softmax}_j(\mathrm{x}_i^T W_q W_k \mathrm{x}_j / \sqrt{128})$$

and x_i and x_j are the embeddings of tokens i and j from prior layer

Multiheaded attention

96 parallel attention heads
Think of each computing a different representation
Followed by a Feedforward (1 hidden layer)



(Image source: GPT architecture on a napkin)

GPT-3

GPT-3 has 96 layers as above

layers also have dropouts

Parameters (estimate)

Embedding: 50527×12288

GPT-3

GPT-3 has 96 layers as above
    layers also have dropouts

Parameters (estimate)
Embedding: 50527×12288
Attention
    96 parallel heads
        Not counting dropouts, biases, layer norm scalings
        Each attention head: $12288 \times 128 \times 3$
    Layer pooling $128 \times 96 \times 12288 = 12288 \times 12288$

GPT-3

GPT-3 has 96 layers as above

  layers also have dropouts

Parameters (estimate)

Embedding: 50527×12288

Attention

  96 parallel heads

    Not counting dropouts, biases, layer norm scalings

    Each attention head: $12288 \times 128 \times 3$

  Layer pooling $128 \times 96 \times 12288 = 12288 \times 12288$

MLP: $12288 \times (4 \times 12288) \times 2$

GPT-3

GPT-3 has 96 layers as above

  layers also have dropouts

Parameters (estimate)

Embedding: $50527\times$ 12288

Attention

  96 parallel heads

    Not counting dropouts, biases, layer norm scalings

    Each attention head: $12288\times128\times3$

  Layer pooling $128\times96\times12288{=}12288\times12288$

MLP: $12288\times(4\times12288)\times2$

$96\times(\text{Attention}+\text{MLP})$

  $= 96\ \times\ (12288\times128\times3\times96+12288\times12288\times9)$

Total: 174.6 billion parameters, (reported 175 billion)

What happens at each layer

Think of each layer as a representation of token
First layer: direct embedding
Subsequent layers: contextualized embeddings
    Richer representation that includes context

What can we do with these rich representations?

Downstream tasks

We have been talking about:
Contextual representation $\rightarrow$ Language model

But in fact, lot lot more

- Translation
- Summarization
- General Knowledge Q&A
- Chatbots
- Programming... and the list goes on

LLMs are few-shot learners

Two general ways to build

LLMs are few-shot learners

Two general ways to build

Fine tuning:

- Uses 1000s/100,000 more examples
- Gradient updates are performed on model
- Original LLMs or subset or (likely) add-on

LLMs are few-shot learners

Two general ways to build

Fine tuning:

Uses 1000s/100,000 more examples

Gradient updates are performed on model

Original LLMs or subset or (likely) add-on

Few shot learning: no parameter updates

Few examples, 10s

(whatever fits into 2048 tokens)

No gradient updates

Use off the shelf predictions

Few shot/in context learning

Perhaps the most striking novel behavior

A pretrained model T seems to learn novel patterns without any weight updates

Demo

In context learning

Mechanism of in-context learning not completely clear

In context learning

Mechanism of in-context learning not completely clear

Many hypothesis

Large and small transformers may do it differently

In context learning

Mechanism of in-context learning not completely clear

Many hypothesis

- Large and small transformers may do it differently
- Transformers may do ICL different from other architectures

In context learning

Mechanism of in-context learning not completely clear

Many hypothesis

- Large and small transformers may do it differently
- Transformers may do ICL different from other architectures
- Perhaps mimic Bayesian predictors

Digging deeper

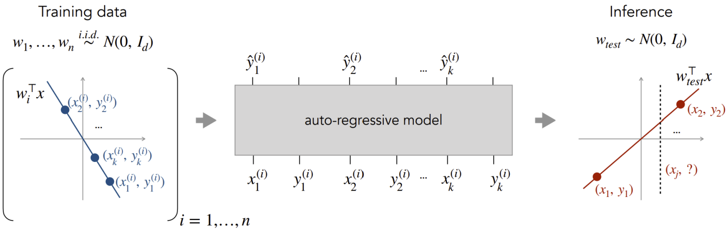


Image:

Garg et. al. 2022

Digging deeper

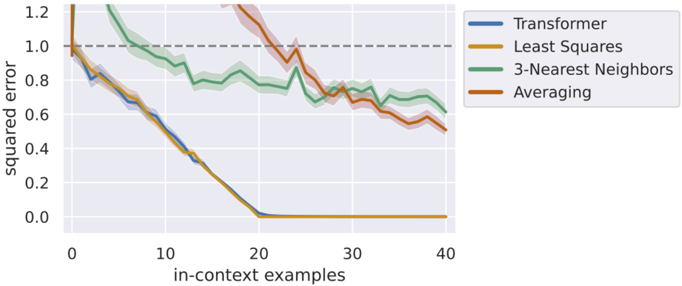
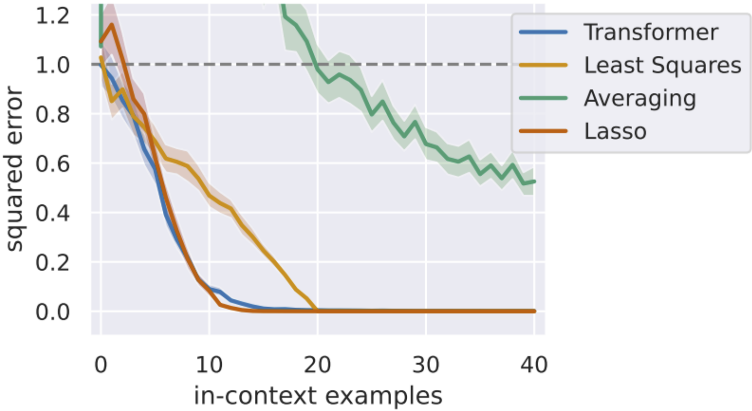


Image: Garg et. al. 2022

University of Hawai'i Mānoa

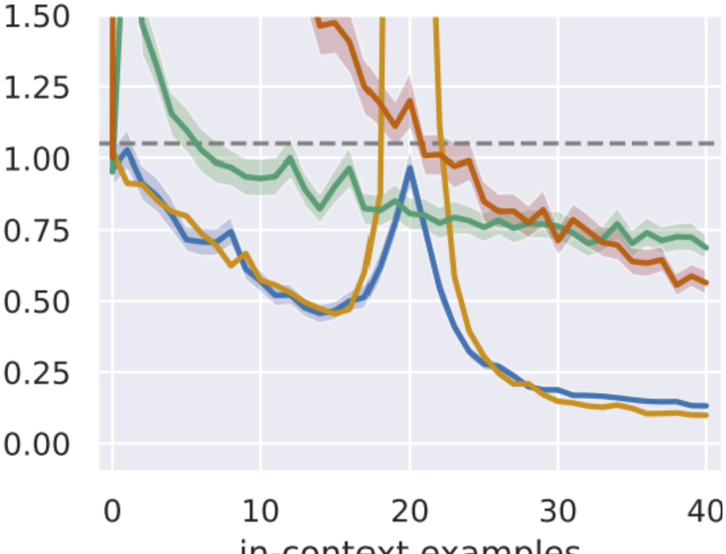
Digging deeper



(a) Sparse linear functions

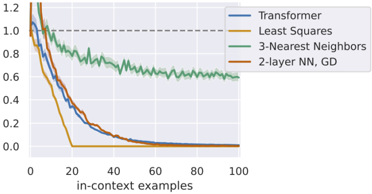
Image: Garg et. al. 2022

Digging deeper



Digging deeper

(c) 2-layer NN



(d) 2-layer NN, eval on linear functions

Image: Garg et. al. 2022

Digging deeper

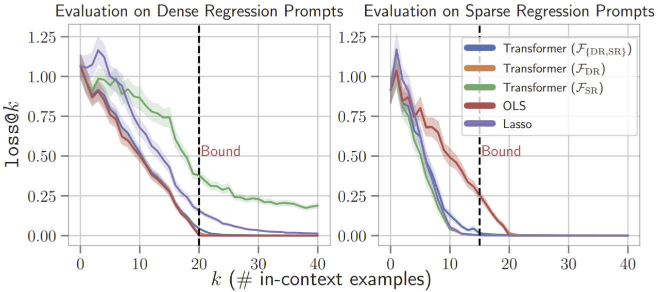


Image: Ahuja et. al. 2023, Bayesian Prism