Natural Language Processing

Narayana Santhanam

EE 645

Mar 25, 2023

This module

Latent Semantic Indexing

- SVD

Language models (Transformers)

- Low rank projections
- Transfer of information

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

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

$\mathbf{u}_i$ all have length 1, mutually perpendicular

cols of V: basis for rows of M

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

$\mathbf{v}_i$ all have length 1, mutually perpendicular

singular values: importance of basis vectors

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

Multiplying out

M is $n \times p$,

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

Instructive to multiply out:

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

Multiplying out

M is $n \times p$,

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

Instructive to multiply out:

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

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

Multiplying out

M is $n \times p$,

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

Instructive to multiply out:

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

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

Multiplying out

M is $n \times p$,

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

Instructive to multiply out:

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

Each of $\mathbf{u}_i \mathbf{v}_i^T$ 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 $\sigma_1 \geq \sigma_2 \geq ...$)

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

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

Latent Semantic Indexing

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

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

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

Latent Semantic Indexing

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

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

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

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

UNIVERSITY of HAWAI'I® MĀNOA

Modern Language Models

Tokenizer (▸ OpenAI)

Brief history:

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

Transformers

What is a transformer?

Central to Transformers is the notion of *attention*

Attention-like approaches in

- Linear Regression
- Kernels

Transformer core

Attention

Skip connections

Attention-like approaches

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

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

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

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

Attention

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

Other algorithms

Ridge Regression

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

Other algorithms

Ridge Regression

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

Support vector machines

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

Soft prediction

$$\mathbf{z}^T\hat{\mathbf{w}} = \sum_{i=1}^{n}\beta_i(\mathbf{z}^T\mathbf{x}_i)\mathbf{y}_i$$

β_i is obtained by solving the dual, most are 0

Other algorithms

Ridge Regression

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

Support vector machines

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

Soft prediction

$$\mathbf{z}^T\hat{\mathbf{w}} = \sum_{i=1}^{n}\beta_i(\mathbf{z}^T\mathbf{x}_i)\mathbf{y}_i$$

β_i is obtained by solving the dual, most are 0

Attention

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

Attention

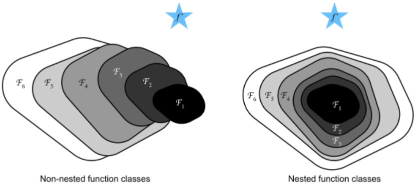
We specialize the observation in prior slides

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

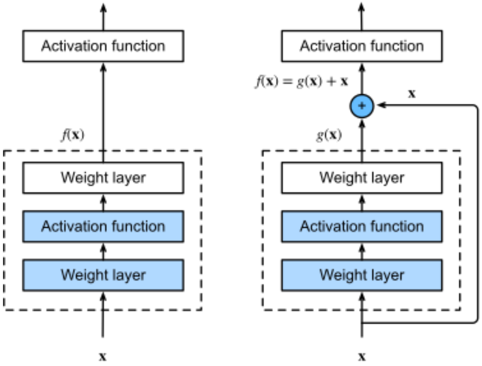
Arbitrary function and pass it through softmax

Skip connections



(Image source: Dive into deep learning)

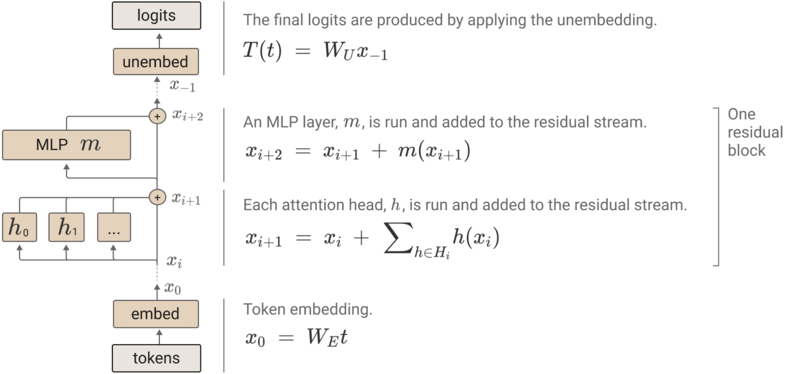
Skip connections



(Image source: Dive into deep learning)

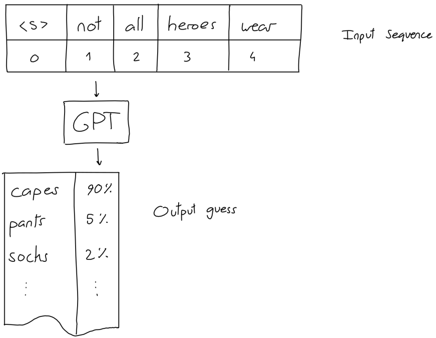
University of Hawai'i Mānoa

Putting them together



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

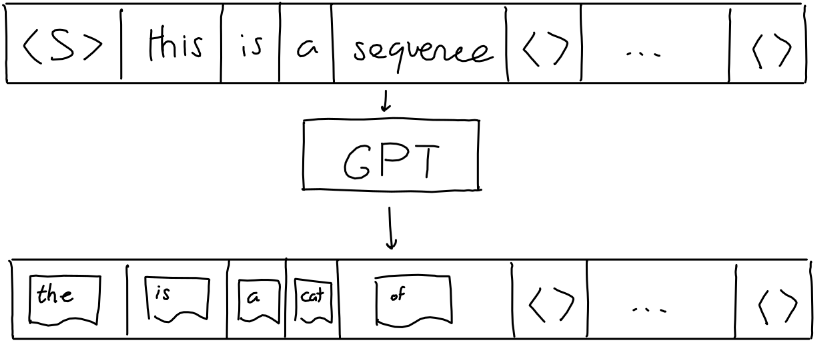
What is a Language Model?



(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

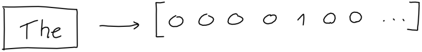
GPT has a vocabulary of 50,257 tokens

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

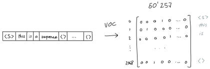
For every token in context

Representation of tokens

GPT has a vocabulary of 50,257 tokens

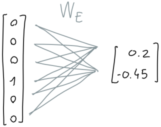


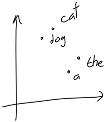
For every token in context



(Image source: GPT architecture on a napkin)

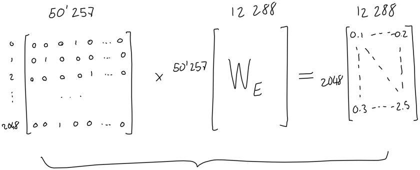
Embeding tokens

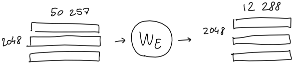




Linear map

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





Positional Encoding

Each position (0-2047) $\to \mathbb{R}^{12288}$
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)

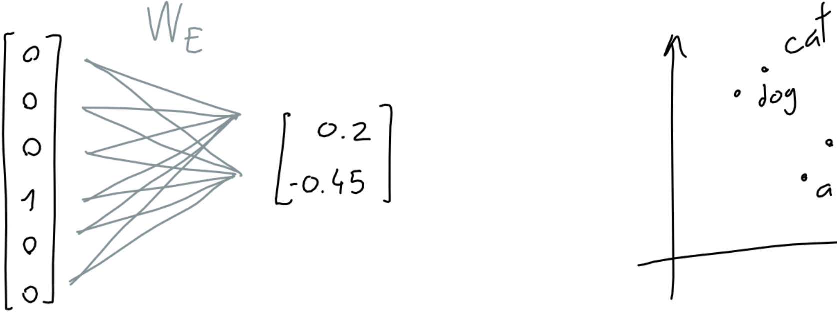
Positional Encoding

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

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

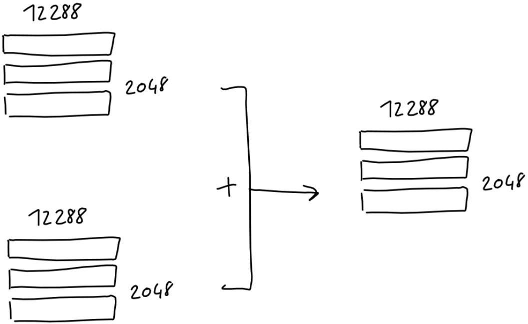
M is a large number (not important)
Idea: mimic binary representation of numbers
relative location is a linear transform

Positional encoding matrix P



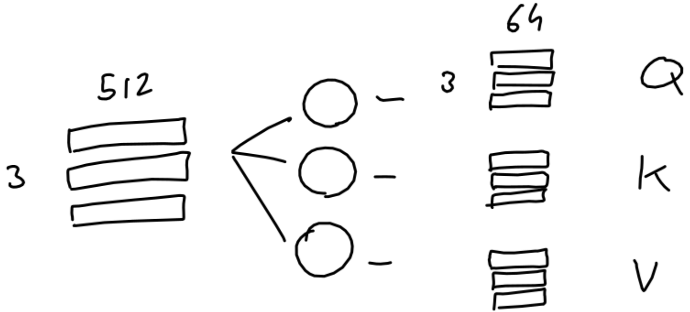
(Image source: Dive into Deep Learning)

Embedding all 2048 tokens



(Image source: GPT architecture on a napkin)

Transformer core: attention



In GPT-3: query, key, values are 128-long vectors

(Image source: GPT architecture on a napkin)

Transformer core: attention

Compute softmax$((QK^T)V)$
For query $\mathbf{q}_i$ from token i, compute

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

for every key $\mathbf{k}_j$ and value $\mathbf{v}_j$ from token j,

Transformer core: attention

Compute softmax$((QK^T)V)$
For query $\mathbf{q}_i$ from token i, compute

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

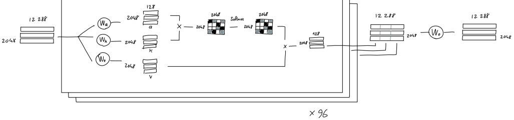
for every key $\mathbf{k}_j$ and value $\mathbf{v}_j$ from token j,

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

and $\mathbf{x}_i$ and $\mathbf{x}_j$ 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: $12288\times128\times3$

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

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

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: $12288\times128\times3$

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