Transformers

Transformers map sequences of vectors to sequences of vectors — in contrast to standard NNs, which map a fixed vec to fixed vec.

Input: Sequence of N "tokens" $x_n^{(0)}$ of dimension D.
Collected into input matrix $X^{(0)} \in \mathbb{R}^{D \times N}$. $X^{(0)} = \left(x_1^{(0)} \cdots x_N^{(0)} \right)$

- Image: pixels are vectors in $[256]^3$. You can imagine taking 4×4 patches, each patch has $16 \cdot 3$ elements in $\{0, 1, \ldots, 255\}$. Stack these as columns
- Language: text can be broken up into words ("tokens"), each word has a fixed vector which represents it. eg., for "vocab" of V words, could use one-hot encoding for each word/token in the vocab. Could learn embeddings, or keep fixed.
- Supervised learning: if you have (x_i, y_i) pairs, you could let tokens be $\begin{pmatrix} x_i \\ y_i \end{pmatrix} \in \mathbb{R}^{d+1}$.

Transformer blocks:

- Each layer is mapped to next layer through "transformer block":

$$X^{(m)} = \text{xformer-block}(X^{(m-1)}) \in \mathbb{R}^{D \times N}.$$

- Two main components:
  - Interactions between tokens: generally requires $N^2 d$ time; allows for understanding how similar/dissimilar i^{th} & j^{th} tokens are
  - Per-token feature refinement: apply nonlinear NN to token's representation

Stage 1: self-attention. We will map $X^{(m)} \longrightarrow Y^{(m)} \in \mathbb{R}^{D \times N}$ $\longrightarrow$ columns $y_n^{(m)}$.

$Y^{(m)} = X^{(m-1)} A^{(m)} \Rightarrow y_n^{(m)} = \sum_{n'=1}^{N} x_{n'}^{(m)} A_{n',n}^{(m)}$, where

- $A_{n',n}^{(m)} \in \mathbb{R}^{N \times N}$ is attention matrix, satisfies $\sum_{n'=1}^{N} A_{n',n}^{(m)} = 1$, $A_{n',n} \geq 0$,
- large vals of $A_{n',n}^{(m)} \leftrightarrow$ n' token highly relevant to n token.
- $y_n^{(m)}$ is convex combo of tokens $\{x_i^{(m)}\}$, weighted by attn.

Self-attn uses the $x_i^{(m)}$ to generate $A^{(m)}$.

Many variants of attn. One simple one:

$$A_{n,n'} := \frac{\exp(\langle x_n, x_{n'}\rangle)}{\sum_{l=1}^{N} \exp(\langle x_n, x_l\rangle)}$$

→ each dot product of D dim vectors.
→ N^2 entries ⇒ $N^2 D$ time.

A more flexible form of attn would be to introduce learnable matrix $U \in \mathbb{R}^{K \times D}$ so that

$$A_{n,n'} = \exp(\langle Ux_n, Ux_{n'}\rangle) \Big/ \sum_{l=1}^{N} \exp(\langle Ux_{n'}, Ux_l\rangle)$$

→ Computing $Ux_n \in \mathbb{R}^K$ requires computing K dot products of D dim vecs; KD time; compute for all N vecs, NKD time

→ then computing N^2 dot prods of K dim vectors, $N^2 K$ time.

→ total of $NKD + N^2K$ time, $N^2 D$ if $K < D/2$

Similarly could use <u>two</u> learnable matrices U_q, U_k & compute

$$A_{n,n'} = \exp(\langle U_k x_n, U_q x_{n'}\rangle) \Big/ \sum_{l}^{N} \exp(\langle U_k x_{n'}, U_q x_l\rangle).$$

$q_n := U_q x_n$, $k_n := U_k x_n$ are called <u>query</u> & <u>key</u> vectors, resp.

Multi-head self attention (MHSA):

- Above allows for one notion of similarity — $\langle U_k x_n, U_q x_{n'} \rangle$.
  Would be better to allow for multiple similarity metrics.
- Introduce H **heads**: each "head" has params $U_{q,h}^{(m)}, U_{k,h}^{(m)}, U_{v,h}^{(m)} \in \mathbb{R}^{K \times D}$, $U_{p,h}^{(m)} \in \mathbb{R}^{D \times K}$

$$q_{h,n}^{(m)} := U_{q,h}^{(m)} x_n^{(m-1)}, \quad k_{h,n}^{(m)} := U_{k,h}^{(m)} x_n^{(m-1)},$$

$$[A_h^{(m)}]_{n,n'} := \exp(\langle k_{h,n}^{(m)}, q_{h,n'}^{(m)} \rangle) \Big/ \sum_{\ell} \exp(\langle k_{h,\ell}^{(m)}, q_{h,n'}^{(m)} \rangle)$$

$$Y_h^{(m)} := \underbrace{U_{v,h}^{(m)}}_{K \times D} \underbrace{X^{(m-1)}}_{D \times N} \underbrace{A_h^{(m)}}_{N \times N}; \quad \in \mathbb{R}^{K \times N}$$

$$Y^{(m)} = \mathrm{MHSA}_{\theta}(X^{(m-1)}) = \sum_{h=1}^{H} \underbrace{U_{p,h}^{(m)}}_{D \times K} \underbrace{Y_h^{(m)}}_{K \times N}$$

θ = params. For each MHSA layer, have H heads, each of which has key/query/value/projection matrices

value $\uparrow U_{v,h}$ — projection $\hookrightarrow U_{p,h}$

<u>Second stage</u>: after getting $Y^{(m)}$, we pass each column (token representation) through an MLP, $\tilde{x}_n^{(m)} = MLP_\theta(y_n^{(m)})$.

- MLP_θ is usually a 2-3 layer F.C. network
- Can often dominate computation time in transformers

Two more ingredients before can introduce full Xformer block.

① Residual connection ② Token normalization.

① we parameterize $x^{(m)} = x^{(m-1)} + res_\theta(x^{(m-1)}) \rightarrow$ just learning $x^{(m)} - x^{(m-1)}$.

② Layer norm is default technique.

$$[x_n]_\ell \mapsto \frac{1}{\sqrt{var(x_n)}}([\bar{x}_n]_\ell - mean(x_n)), \quad mean(x_n) = \frac{1}{D}\sum_1^D [x_n]_\ell,$$

$$var(x_n) = \frac{1}{D}\sum_1^D ([x_n]_\ell - mean(x_n))^2.$$

Final recipe:

$$x^{(m-1)} \mapsto LayerNorm(x^{(m-1)}) = \bar{X}^{(m-1)} \mapsto Y^{(m)} = x^{(m-1)} + MHSA(\bar{X}^{(m-1)}) \mapsto \bar{Y}^{(m)} = LayerNorm(Y^{(m)})$$

$$\downarrow$$

$$X^{(m)} = Y^{(m)} + MLP(\bar{Y}^{(m)})$$

Position encoding:

- Transformer as we've described doesn't depend on order of tokens
- OK in some contexts, not for language
- The fix is to use "position encoders": if ~~token~~ appears in i^{th} position, we add to token embedding $x_n^{(0)}$ some value $p_n^{(0)}$.
- eg in language, input to transformer is,

$$x_n^{(0)} = \underbrace{e_n^{(0)}}_{\text{embedding corresp. to token}} + p_n^{(0)}$$

embedding corresp to "n^{th}" token in seq.

Since transformers can map sequences to sequences, they can operate as learning algorithms at test time:
after being trained, they can take as input (x_i, y_i) sequences, then formulate predictions for query examples.

We'll now discuss one setting where we can understand this behavior precisely:

Linear transformers trained on random linear regression tasks.

Model: A single layer, single-head softmax-based attn has form, for input matrix $E \in \mathbb{R}^{D \times N}$, (no normalization layer)

$$f(E; W^K, W^Q, W^V, W^P) = E + W^P W^V E \operatorname{softmax}\left(\frac{(W^K E)^T W^Q E}{\rho}\right)$$

ρ: normaliz. factor

We will look at following simplified model: for $\theta = (W^{PV}, W^{KQ})$,

$$f_{LSA}(E;\theta) = E + W^{PV} E \cdot \frac{E^T W^{KQ} E}{\rho},$$

Linear Self Attention: no softmax. Also merged W^K, W^Q & W^P, W^V.

Setting: We suppose we have data as follows. Let $\Lambda \succ 0$, $\Lambda \in \mathbb{R}^{d \times d}$.

- $w^{(\tau)} \overset{iid}{\sim} N(0, I_d)$
- $x_i^{(\tau)} \overset{iid}{\sim} N(0, \Lambda)$
- $y_i^{(\tau)} \overset{iid}{\sim} \langle x_i^{(\tau)}, w^{(\tau)} \rangle$.

Receive datasets

$$D^{(\tau)} = \{(x_i^{(\tau)}, y_i^{(\tau)})\}_{i=1}^{N+1}, \quad \tau = 1, \dots, B$$

Define token embedding matrices

$$E^{(\tau)} = \begin{pmatrix} x_1^{(\tau)} & \cdots & x_N^{(\tau)} & x_{N+1}^{(\tau)} \\ y_1^{(\tau)} & \cdots & y_N^{(\tau)} & 0 \end{pmatrix} \in \mathbb{R}^{d+1 \times N+1}.$$

$f_{LSA}(E^{(\tau)};\theta) \in \mathbb{R}^{d+1 \times N+1}$

Prediction for $y_{N+1}^{(\tau)}$: $[f_{LSA}(E^{(\tau)};\theta)]_{d+1, N+1} =: \hat{y}^{\tau}_{query}$

Define loss function

$$\hat{L}(\theta) = \frac{1}{2B} \sum_{\tau=1}^{B} \Big(\underbrace{\hat{y}^{(\tau)}_{query}}_{\text{comes form } f_{LSA}\text{; depends on } \theta = (W^{KQ}, W^{PV})} - y_{N+1}^{(\tau)} \Big)^2$$

Consider limit as $B \to \infty$: this is "infinite pretraining data" limit. Then,

$$L(\theta) = \lim_{B\to\infty} \hat{L}(\theta) = \frac{1}{2} \mathbb{E}_{w^\tau, x_i^\tau} \left[\left(\hat{y}^\tau_{query} - y^{(\tau)}_{N+1} \right)^2 \right].$$

Let's consider gradient flow on this: $\frac{d\theta}{dt} = -\nabla L(\theta)$.

<u>Theorem</u> [Zhang-Frei-Bartlett JMLR '24]:

For a suitable random init, GF converges to a global min of $L(\theta)$. Moreover, if $\Gamma_N := \left(1+\frac{1}{N}\right)\Lambda + \frac{1}{N}\mathrm{tr}(\Lambda) I_d$, Then the transformer converges to a network θ^* which satisfies, for any $(x_i, y_i)_{i=1}^M$,

$$\underbrace{\left[f_{LSA}\left(\begin{pmatrix} x_1 & \cdots & x_M & x \\ y_1 & \cdots & y_M & 0 \end{pmatrix}; \theta^* \right) \right]_{d+1, M+1}}_{=: \hat{y}_{query}(x)} = x^T \Gamma_N^{-1} \left(\frac{1}{M} \sum_{i=1}^M y_i x_i \right)$$

$$= x^T \left[\left(1+\frac{1}{N}\right)\Lambda + \frac{1}{N}\mathrm{tr}(\Lambda) I_d \right]^{-1} \cdot \left(\frac{1}{M} \sum_1^M y_i x_i \right).$$

<u>Remarks</u>. If $y_i = \langle w, x_i \rangle \ \forall i$, then: $\frac{1}{M}\sum_1^M y_i x_i = \frac{1}{M}\sum_1^M x_i x_i^T w$
$= \left(\frac{1}{M}\sum_1^M x_i x_i^T\right) w.$

- If $N \to \infty$, $\Gamma_N \to \Lambda$, so if ↑ holds too, we get

$$\hat{y}_{query}(x) \to x^T \Lambda^{-1} \cdot \left(\frac{1}{M}\sum_1^M x_i x_i^T \right) w.$$

Γ_N is a regularized version of Λ.

- If x_i are iid, $\frac{1}{M}\sum_1^M x_i x_i^T \to \mathbb{E} x x^T$. If $\mathbb{E} x x^T = \Lambda$, then $\hat{y}_{query}(x) \to x^T w$.

- If x_i are not iid, or if $\mathbb{E}xx^T \neq \Lambda$ (distribution shift), then we should <u>not</u> expect the trained transformer to do well for new linear regression tasks.

What happens if $y_i \neq \langle w, x_i \rangle$?

<u>Theorem</u> (Zhang-Frei-Bartlett) Suppose $(x_i, y_i) \overset{iid}{\sim} D$, where $x_i \sim N(0, \Lambda)$. Suppose $\mathbb{E}y$, $\mathbb{E}xy$, $\mathbb{E}y^2xx^T$ exist & are finite. Then, for $\hat{y}$: prediction for trained transformer, if we let

$$a := \Lambda^{-1}\mathbb{E}xy, \qquad \Sigma := \mathbb{E}\left[(xy - \mathbb{E}xy)(xy - \mathbb{E}xy)^T\right],$$

then:

$$\mathbb{E}\left[(\hat{y} - y)^2\right] = \min_{w \in \mathbb{R}^d} \mathbb{E}\left[(\langle w, x \rangle - y)^2\right] + \frac{1}{M} tr\left(\Sigma \Gamma^{-2} \Lambda\right) + \frac{1}{N^2}\left(\|a\|^2_{\Gamma^{-2}\Lambda^3} + 2\, tr(\Lambda)\|a\|^2_{\Gamma^{-2}\Lambda^2} + tr(\Lambda)^2 \|a\|^2_{\Gamma^{-2}\Lambda}\right)$$

<u>Remarks</u>

- A type of "emergent behavior": trained on <u>noiseless</u> linear regression, but achieves <u>best linear prediction error</u>. So does well on noisy regression, nonlinear, ...
- Training contexts have different effect than test-time: $\frac{1}{N^2}$ vs $\frac{1}{M}$.

Proof ingredients : Fairly involved ...

① Show $L(\theta) \leftrightarrow L(u) = \mathbb{E}\left[(u^T H u - y)^2 \right]$ for some non-psd H, so L is non-cvx.

② Establish PL-inequality:

$$\|\nabla L(u)\|^2 \geq c \cdot \left(L(u) - \min_u L(u) \right)^2 \quad \text{for some } c > 0.$$

This is 90% of proof. Has some analogues to proof ideas in training two-layer linear networks (recall: $f(E;\theta) = E + W^{PV} E \cdot \frac{E^T W^{KQ} E}{\rho}$. Two linear layers, W^{KQ}, W^{PV} appear.)

③ Identify limit of gradient flow

④ Characterize statistical properties of the limit.