Transformers as Statisticians: Provable In-Context Learning with In-Context Algorithm Selection

Song Mei
UC Berkeley

Joint work with
Yu Bai (Salesforce Research), Fan Chen (Peking U -> MIT PhD), Huan Wang (Salesforce Research), Caiming Xiong (Salesforce Research)
Large Language model

- **Task**: next word prediction.
- **Input**: sequence of words. **Output**: the next word.
- **Training data**: texts from the whole internet, e.g., Wikipedia, Reddit, Shakespeare books, etc.
- **Network architecture**: transformers.
An emergent capability: In-context learning (ICL)

Apple -> Fruit. Sofa -> Furniture. Bird -> ?

Animal.

Circulation revenue has increased by 5% in Finland.
Positive.
Tom didn't disclose the purchase price.
Neutral.
Paying off the national debt will be extremely painful.

Negative.

Supervised learning vs In-context learning

Dataset: \{ (x_i, y_i) \}_{i \in [N]}, (x_{N+1}, y_{N+1}) \sim \mathbb{P}

- **Supervised learning:**
  1. Train a model \( y = f(x; \hat{w}) \) using the training set \( \{(x_i, y_i)\}_{1 \leq i \leq N} \).
  2. Output prediction \( \hat{y}_{N+1} = f(x_{N+1}; \hat{w}) \approx y_{N+1} \)

- **In-context learning:**
  1. Pre-train a model \( h = f(H; \hat{\theta}) \) using an enormous meta-dataset.
  2. Take \( H = [x_1, y_1, x_2, y_2, \ldots, x_N, y_N, x_{N+1}] \) as the context input.
  3. Output prediction \( \hat{y}_{N+1} = f(H; \hat{\theta}) \approx y_{N+1} \).

<table>
<thead>
<tr>
<th>Apple -&gt; Fruit.</th>
<th>Sofa -&gt; Furniture.</th>
<th>Bird -&gt; ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 ) ( y_1 )</td>
<td>( x_2 ) ( y_2 )</td>
<td>( x_3 )</td>
</tr>
<tr>
<td>Animal. ( \hat{y}_3 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An ICL experiment on a synthetic dataset

A Task:  \[ \{(x_i, y_i)\}_{i \in [N]}, \quad \beta \sim \mathcal{N}(0, I_d/d), \]
\[ x_i \sim \mathcal{N}(0, I_d), \quad y_i = x_i^T \beta + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2) \]

- A dataset of (size \( N \)) is a meta-datapoint: \( H = [x_1, y_1, x_2, y_2, \ldots, x_N, y_N] \).
- A meta-dataset (size \( n \)): \( \{H^{(j)} = [x_1^{(j)}, y_1^{(j)}, x_2^{(j)}, y_2^{(j)}, \ldots, x_N^{(j)}, y_N^{(j)}]\} \) \( j \in [n] \).
- Train the GPT2 model using \( \{H^{(j)}\} \) \( j \in [n] \) (a smaller version of ChatGPT).
- \( d = 5, \quad N = 40, \quad n = 19,200,000 \)
- Evaluate the test performance of GPT2 on a new independent task.

- Trained GPT2 performs as good as Bayesian predictor!
Why can GPT (transformers) perform in-context learning (ICL)?

[Bai, Chen, Wang, Xiong, Mei, 2023].

Related work: [Xie et al., 2021], [Garg et al. 2022], [Akyurek et al., 2022], [von Oswald et al., 2022], [Dai et al., 2022]
Transformers

- A transformer* is a sequence-to-sequence neural network $\text{TF}_\theta : \mathbb{R}^{D \times N} \rightarrow \mathbb{R}^{D \times N}$.
- Input sequence: $H = [h_1, h_2, \ldots, h_N] \in \mathbb{R}^{D \times N}$; each $h_i \in \mathbb{R}^D$ is called a token.

$$H = \begin{bmatrix} h_1 & h_2 & h_3 & \vdots & \vdots & h_N \end{bmatrix}$$

$$\text{TF}_\theta$$

$$H' = \begin{bmatrix} h'_1 & h'_2 & h'_3 & \vdots & \vdots & h'_N \end{bmatrix}$$

read

$\hat{y}$

- Transformer output: $H' = \text{TF}_\theta(H) = [h'_1, \ldots, h'_N] \in \mathbb{R}^{d \times N}$.
- Final output: $\hat{y} = \text{read}(\text{TF}_\theta(H))$.

* Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017). We focus on the encoder architecture.
Transformers: Feedforward layer

- A transformer is an iterative composition of FF layers and Attention layers

$$\text{TF}_\theta(\cdot) = \text{FF}_{W^{(L)}} \circ \text{ATTN}_{A^{(L)}} \circ \cdots \circ \text{FF}_{W^{(1)}} \circ \text{ATTN}_{A^{(1)}}$$

$$\theta = (W^{(L)}, A^{(L)}, \ldots, W^{(1)}, A^{(1)})$$

- Feedforward layer: $H' = \text{FF}_{W}(H) : \mathbb{R}^{D \times N} \to \mathbb{R}^{D \times N}$,

$$W = (W_1, W_2), \quad W_1, W_2^\top \in \mathbb{R}^{D' \times D}$$

- Token-wise function:

$$h'_i = h_i + W_2 \cdot \sigma(W_1 h_i)$$

Transformers: Attention layer

- A transformer is an iterative composition of FF layers and Attention layers

\[ TF_\theta(\cdot) = \text{FF}_{W(L)} \circ \text{ATTN}_{A(L)} \circ \cdots \circ \text{FF}_{W(1)} \circ \text{ATTN}_{A(1)} \]

\[ \theta = (W(L), A(L), \ldots, W(1), A(1)) \]

- Attention layer: \( H' = \text{ATTN}_A(H) : \mathbb{R}^{D \times N} \to \mathbb{R}^{D \times N} \)

\[ A = (Q_m, K_m, V_m)_{m \in [M]}, \quad Q_m, K_m, V_m \in \mathbb{R}^{D \times D} \]

- Single-head attention layer:

\[
h'_i = h_i + \sum_{M=1}^{M} \sum_{j=1}^{N} \sigma(Q_k h_i K_k h_j) V_k h_j
\]

Transformers: the whole structure

![Diagram of Transformer architecture]

- **BERT (2018):** $L = 24$, $M = 12$, $D = 1024$, #para = 340M
- **GPT2 (2019):** $L = 48$, $M = 25$, $D = 1600$, #para = 1.5B
- **GPT3 (2020):** $L = 96$, $M = 96$, $D = 12288$, #para = 175B
- **PALM (2022):** $L = 118$, $M = 48$, $D = 11432$, #param = 540B

In-context learning (ICL) by Transformers
**ICL by transformers**

\[
\{(x_i, y_i)\}_{i \in [N+1]} \sim_{iid} P, \quad y_i = f(\langle x_i, w_\ast \rangle) + \epsilon_i
\]

- **Empirical observation:** on pre-trained Transformers, \(\mathbb{E}\ell(\hat{y}_{N+1}^{\text{TF}}, y_{N+1})\) is small.

- **Explanation:** transformers can approximate GD on \(\hat{R}_N(w) = \frac{1}{N} \sum_{i=1}^{N} \ell(x_i^Tw, y_i)\)

\[
w^{t+1} = w^t - \frac{\eta}{N} \sum_{i=1}^{N} \partial_1 \ell(x_i^Tw^t, y_i) \times x_i
\]

[Akyurek et al., 2022], [von Oswald et al., 2022], [Dai et al., 2022] have rough ideas of this kind.
One-step gradient descent by an attention layer

\[
\frac{\partial}{\partial \ell_1} \approx \sum_{m=1}^M a_m \cdot \sigma \left( b_m \left\langle x_i, w^t \right\rangle + c_m \cdot y_i \right)
\]

\[
\frac{w^t}{\tau} = \frac{\eta}{N} \sum_{i=1}^N \frac{\partial}{\partial \ell} \left( \left\langle x_i, w^t \right\rangle, y_i \right) \times x_i
\]

Attention

Weight

Universal approximation

Gradient descent
Transformer versus multi-step GD

\begin{align*}
\text{Transformer} & \quad \text{Gradient descent} \\
\text{Input} & \quad w^0 = 0 \\
\text{ATTN}_{A^{(1)}} & \quad w^1 = w^0 - \frac{\eta}{N} \sum_{i=1}^{N} \partial_1 \ell(x_i^T w^0, y_i) \times x_i \\
\text{ATTN}_{A^{(2)}} & \quad w^2 = w^1 - \frac{\eta}{N} \sum_{i=1}^{N} \partial_1 \ell(x_i^T w^1, y_i) \times x_i \\
\text{ATTN}_{A^{(L)}} & \quad w^L = w^{L-1} - \frac{\eta}{N} \sum_{i=1}^{N} \partial_1 \ell(x_i^T w^{L-1}, y_i) \times x_i \\
\text{ATTN}_{A^{(L+1)}} & \quad \hat{y}_{N+1} = f((x_{N+1}, w^L)) \\
\text{TF}_\theta & \quad \text{read} \\
\hat{y} & \quad \text{read} \\
\end{align*}
Transformer versus proximal gradient descent

Transformer

- Input
- \text{ATTN}_{A^{(1)}}
- FF_{W^{(2)}}
- FF_{W^{(L)}}
- \text{ATTN}_{A^{(L+1)}}
- TF_{\theta}

Proximal gradient descent

\[
\begin{align*}
    \text{Input} & \quad h_1 \quad h_2 \quad h_3 \quad \vdots \quad h_{N'} \\
    \text{ATTN}_{A^{(1)}} & \quad h_1 \quad h_2 \quad h_3 \quad \vdots \quad h_{N'} \\
    \text{FF}_{W^{(2)}} & \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
    \text{FF}_{W^{(L)}} & \quad h_1 \quad h_2 \quad h_3 \quad \vdots \quad h_{N'} \\
    \text{ATTN}_{A^{(L+1)}} & \quad h_1 \quad h_2 \quad h_3 \quad \vdots \quad h_{N'} \\
    \text{TF}_{\theta} & \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
    \text{read} & \quad \hat{y} \\
\end{align*}
\]

\[
\begin{align*}
    \text{Proximal gradient descent} \\
    w^0 &= 0 \\
    w^{0.5} &= w^0 - \frac{\eta}{N} \sum_{i=1}^{N} \partial_1 \ell(x_i^T w^0, y_i) \times x_i \\
    w^1 &= \text{Prox}_{\eta R}(w^{0.5}) \\
    w^L &= \text{Prox}_{\eta R}(w^{L-0.5}) \\
    \hat{y}_{N+1} &= f((x_{N+1}, w^L)) \\
\end{align*}
\]
In-context GD

Important parameters:
Embedding dimensions $D$, number of layers $L$, number of ATTN heads $M$, FF width $D'$, and norm of parameters $\|\theta\|$

**Thm [In-context gradient descent]**: There exists a transformer with

$D = \Theta(d), L = \Theta(1/\eta), M = M(\epsilon), D' = \Theta(d), \|\theta\| = \Theta(\text{Poly}(D, L, M, D', N)),$

that output $\hat{w}^L$ that is close to the GD iterates $w^L_{\text{GD}}$

$$\|\hat{w}^L - w^L_{\text{GD}}\|_2 \leq L\eta\epsilon.$$
**Thm:** There exists three transformers with $D = D' = \mathcal{O}(d)$, $\|\theta\| = \mathcal{O}(\text{Poly}(\cdot))$

- **Ridge:** $L = \mathcal{O}(\log(N))$, $M = 3$,
- **Logistic:** $L = \mathcal{O}(\log(N/d))$, $M = N/d$,
- **LASSO:** $L = \mathcal{O}(1 + d/N)$, $M = 2$,

that output $\hat{y}_{N+1}$ implementing **Ridge, Logistic, LASSO**, with

- **Ridge:** $\mathbb{E}[(\hat{y}_{N+1} - y_{N+1})^2] \leq \mathcal{O}(d/N)$,
- **Logistic:** $\mathbb{E}[(\hat{y}_{N+1} - \mathbb{E}[y_{N+1} | x_{N+1}])^2] \leq \mathcal{O}(d/N)$,
- **LASSO:** $\mathbb{E}[(\hat{y}_{N+1} - y_{N+1})^2] \leq \mathcal{O}(s \log d/N)$.
Generalization bound for pre-training

Setting of pre-training:
Meta-dataset \( \{Z^{(j)}\}_{j \in [n]} \sim iid \ \pi \), with \( Z^{(j)} = \{(x^{(j)}_i, y^{(j)}_i)\}_{i \in [N]} \ iid. \)

Form empirical risk \( \hat{L}_{icl}(\theta) = \frac{1}{n} \sum_{j=1}^{n} \ell_{icl}(Z^{(j)}, \theta) \). Consider TF class

\[ \Theta_{L,M,D',B} = \left\{ \theta : L \text{ layers, } M \text{ heads, } D' \text{ width, } B \text{ norm} \right\} \]

**Thm [Generalization for pre-training]:** The generalization bound gives

\[
\sup_{\theta \in \Theta_{L,M,D',B}} \left| \hat{L}_{icl}(\theta) - \mathbb{E}[\hat{L}_{icl}(\theta)] \right| \leq \tilde{O} \left( \sqrt{\frac{L^2(MD^2 + DD')\log B}{n}} \right).
\]

For example, if \( \pi \) is sparse linear model (LASSO), the overall error with \( N \) in-context sample and \( n \) meta-training samples gives

\[
L_{icl}(\hat{\theta}) - \sigma^2 = \tilde{O} \left( \sqrt{\frac{d^4}{n} + \frac{s \log d}{N}} \right)
\]
Simulations

Comparison of ICL error of pre-trained transformers. Parameters $D = D' = 64, L = 12, M = 8$, ReLU attention. Use Adam with learning rate $1e - 4$, batch size 64 for 300k steps.
More surprising experiments
A surprising experiment

- **Two meta-tasks:**
  1. Regression tasks $\hat{\pi}_{\text{reg}}: \beta \sim \mathcal{N}(0, I_d/d), \quad y_i = x_i^\top \beta$.
  2. Classification tasks $\hat{\pi}_{\text{cls}}: \beta \sim \mathcal{N}(0, I_d/d), \quad y_i = \text{Logit}(x_i^\top \beta)$.

- **Three Transformers (same architecture but different pre-training):**
  1. Train TF\_reg using $\{Z_{\text{reg}}^{(j)}\} \sim iid \hat{\pi}_{\text{reg}}$.
  2. Train TF\_cls using $\{Z_{\text{cls}}^{(j)}\} \sim iid \hat{\pi}_{\text{cls}}$.
  3. Train a TF\_unnamed using mixture $\{Z_{\text{reg}}^{(j)}\} \sim iid \hat{\pi}_{\text{reg}}$ and $\{Z_{\text{cls}}^{(j)}\} \sim iid \hat{\pi}_{\text{cls}}$.

- As demonstrated:
  - TF\_reg performs linear regression.
  - TF\_cls performs logistic regression.

What algorithm does TF\_unnamed perform?
Algorithm selection capability of pre-trained TF

- We give \texttt{TF\_unnamed} a name \texttt{TF\_alg\_select}.

\begin{itemize}
  \item \texttt{TF\_alg\_select} selects a proper algorithm on any task.
\end{itemize}

This is what statisticians do in data analysis
Another surprising experiment

Linear model with small noise

Linear model with large noise

- **TF_alg_select** selects ridge regression with optimal regularization.

This is what statisticians do in data analysis.
How statisticians perform algorithms selection?

Two strategies:

- **Strategy 1**: Run \( K \) algorithms in parallel. Select the algorithm with the smallest validation error.
- **Strategy 2**: Perform a hypothesis test to select the algorithm.

Transformers efficiently implement:

- **Mechanism 1**: Post-ICL validation.
- **Mechanism 2**: Pre-ICL testing.
Post-ICL validation mechanism

There exists a transformer that performs the following:

1. Run $K$ different algorithms on $\mathcal{D}_{\text{train}}$, using $L$ steps GD with $L$ layers TF.
2. Choose the prediction function that minimizes the loss on the validation set $\mathcal{D}_{\text{val}}$, and predict $\hat{y}_{N+1}$. This uses in total 3 layers.

**Thm** [Select optimal ridge regularization; informal]: For Bayesian linear model with $K$ different noise levels, there exists a transformer that uses the post-ICL validation mechanism to efficiently implement the ridge regression with optimal regularization parameter. Such a transformer can be pre-trained efficiently.
Pre-ICL testing mechanism

\[
H = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_N & x_{N+1} \\ y_1 & y_2 & y_3 & \cdots & y_N & 0 \end{bmatrix}
\]

Binary test

- There exists a transformer that performs the following:
  1. Perform a simple hypothesis test on the dataset (binary test/correlation test, etc) and decide which algorithm to choose. This uses constant depth TF.
  2. Run the selected algorithm on the whole dataset, and predict \( \hat{y}_{N+1} \).

**Thm [Select regression vs classification; informal]:** There exists a TF that uses the pre-ICL testing mechanism to efficiently implement linear regression on regression tasks, and implement logistic regression on classification tasks. Such a transformer can be pre-trained efficiently.
Summary

• Transformers can efficiently implement basic ICL algorithms using the gradient descent mechanism.

• Transformers can efficiently implement algorithm selection (pre-ICL validation, post-ICL testing), similar to a statistician.

• Such transformers can be pre-trained statistically efficiently.

• Paper will come out this week.