The road to LLMs:
What computational linguists have learned from four decades of experimental statistics

Micha Elsner
Department of Linguistics
THE OHIO STATE UNIVERSITY
I am not a statistician (except when I have to be)

An old proverb says:

   **In theory there is no difference between theory and practice; in practice there is.**

LLMs are remarkably effective… perhaps more so than they have any right to be, given my field’s relatively limited understanding of why they work.

This is all the more true for me personally; there are some excellent statisticians in Comp Ling/Natural Language Processing, but I am not one of them.

I will do my best to lay out the issues, but do please bear with me.
What is a language model?

Imagine some linguistic context (English wikipedia articles…) in which language occurs.

Treat the sequence of words as a sequence of random variables $X_i$

A language model (LM) estimates the probability distribution over the next word:

$$P(X_i | X_0…i-1)$$
A good LM has impressive capabilities

In principle, an LM needs to know an awful lot about language use in context, which also requires:

**Factual knowledge:** *The 16th president of the United States was* (*Abraham*)

**Computation:** *Ten plus two is* (*twelve*)

**Sound systems:** *A word beginning with b and rhyming with “air” is* (*bear*)

**Discourse:** *Mary infuriated John. (She > He):* Kehler and Rohde 2017

**Social expectation:** *The girl will taste the* (*sweets > beer):* Altmann and Kamide 1999
LMs as application components

Shannon’s **noisy channel model** decomposes a task (such as French to English machine translation) into:

\[ P(E|F) \propto P(E)P(F|E) \]


This decomposition was critical in speech recognition as well.

Interest in making this work spurred early research into effective LMs.
Zipf’s law: power-law dynamics

Frequency of words in a text corpus (Canadian parliament)

Logarithmic frequencies
Issue: the statistics of natural language are sparse

Most word sequences of any real length will have never occurred in training data. This makes estimating the probability of continuations difficult.

In order to learn anything useful, we will have to aggregate statistics:

- Across words that bear some sort of similarity (sweets ≈ candy)
- Across long contexts with a shared suffix, hoping that the dynamics are local (whether or not you want to eat this pizza ≈ someone here will eat this pizza)
- Across contexts that share some latent syntactic structure (either Kim knows or ≈ either Robin has gone to the store or)
Ceiling on the performance of simple models

Models with “nice” estimators (identifiable, convex objective) based on manually-designed grouping strategies:

- Group words into clusters (but there are no perfect synonyms)
- Strictly enforce Markov property with time horizon (but long-distance dynamics exist- Chomsky already rejects this class of models in the 1950s)
- Strictly enforce Markov property with respect to latent structure (probably Chomsky-compliant, but trees don’t cover many non-syntactic phenomena, and getting a good distribution over latent parse trees is hard in itself)
Dense models

Represent a word (or sub-word piece) as a real-valued vector.

Sometimes considered geometrically, as projection from \( 1^V \) (sparse) to \( \mathbb{R}^d \) (dense).

Initial objective was to compress/predict word co-occurrence statistics.

Words can share some geometric structure (allowing generalization across semantic or syntactic neighbors) while retaining unique representations.

for example, Latent Semantic Analysis (Deerwester et al. 1988), Word2Vec (Mikolov et al 2013)
Superposition

Embeddings allow generalization across words (*sweets ≈ candy*). We may imagine words having latent, separable features in a very high-dimensional space. (This is conceptually different than the space of word indicator functions we saw on the previous slide.)

The dense model projects multiple features onto the same dimension.

Elhage et al. 2022 “Toy models of superposition”
Superposition leads to complex dynamics

The way in which features are assigned to dimensions changes throughout training.

Reconfigurations produce rapid phase shifts in the loss curve and corresponding sharp changes to the organization of the embedding space.
From vectors to nets

Embeddings allow generalization across words, but not across sequences of words. Doing this requires us to model word-to-word relationships beyond simple co-occurrence.
Models grow deeper and wider

Deep neural net

more layers?

layer 1

layer 2

outputs

inputs

Recurrent neural net

state at time $i$

state at $i+1$

$h1$

$h2$

$h3$

What didn’t work

Datasets were smaller, which limited the potential advantages of these models.

But never mind that, they just didn’t work all that well!

Neural nets are not “nice” models:

  Model is underconstrained: many different optima with similar objective value (some of which have very different behavior)

Stochastic gradient is not all that nice an estimator:

  Gradient dynamics were (and to an extent still are) poorly understood
  Good solutions seemed unreachable from typical starting points
Unstable estimation in a multilayer network

Test error increases (and variance explodes) as number of layers increases from 1-4.

Erhan et al., JMLR 2010
qtd. in Socher et al 2013 tutorial
“Vanishing” gradients

The problem is basic calculus:

If \[ y = F(G(x)) \]
Then \[ y' = F'(G(x))G'(x) \]

In a deep net, gradients of the initial units become a product of increasingly many terms. An early description of this problem: gradients would “vanish” below machine precision if enough of the terms were small, or “explode” if enough of them were large. But even numerically representable gradients won’t necessarily find a good optimum.
Even non-vanishing gradients become noisy

Gradient information becomes increasingly unhelpful as the network grows deeper in both directions (abstraction and sequential time).

Balduzzi et al. "The shattered gradients problem: If resnets are the answer, then what is the question?" (2017)
Long gradient paths connect long-distance dependents

either  Kim  knows  or
Residual networks (ResNets)

Unlike function composition, addition allows gradient terms to separate:

If \[ y = F(x) + G(x) \]
Then \[ y' = F'(x) + G'(x) \]

This is one critical trick for trainable deep networks.

Gradients in ResNets are much more informative

(Skipping some related innovations, like nearly-linear activation functions, batch/layer normalization and adaptive/momentum-based gradient updates.)
Lottery tickets, or, how I learned to stop worrying and love local minima

“A randomly-initialized, dense neural network contains a sub-network that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.”
“the benefit of the initialization is connected to the optimization algorithm, dataset, and model. For example, the winning ticket initialization might land in a region of the loss landscape that is particularly amenable to optimization by the chosen optimization algorithm…

Du et al. (2019) prove that sufficiently overparameterized two-layer relu networks (with fixed-size second layers) trained with SGD converge to global optima. A key question, then, is whether the presence of a winning ticket is necessary or sufficient for SGD to optimize a neural network to a particular test accuracy. We conjecture (but do not empirically show) that SGD seeks out and trains a well-initialized subnetwork.”
Reconsidering temporal dynamics

We’ve seen that mere access to information is not enough to ensure it will be used. Ideally, it should be accessible via a short gradient path (few function compositions or multiplicative operations).

**Attention** mechanisms enable flexible access to the past of a sequence, without requiring separate parameters for every timestep.

\[
c = \sum_i p_i t_i \quad P(t_5 | t_1 \ldots 4) = P(t_5 | c)
\]

circa 2014, in papers like Bahdanau et al. “Neural machine translation by jointly learning to align and translate”
The transformer

Models a word sequence: each token associated with a deep ResNet “column”.

Vaswani et al “Attention is all you need” (2017)
Drawing transverse connections

Sequential effects modeled by attention (between layers only).

Lack of sequential structure within layers makes everything parallel, which is part of how you run it at massive scale.

Vaswani et al “Attention is all you need” (2017)
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:
\[ x_{i+1} = x_i + \sum_{h \in H_i} h(x_i) \]

Token embedding:
\[ x_0 = W_E t \]

Elhage et al 2021 “A Mathematical Framework for Transformer Circuits”
Scaling up

Relatively minor changes and optimizations between:

- Vaswani’s original paper
- BERT (first so-called “foundation model”)
- GPT-3

But significant growth in model size, dataset size and capability.
Scaling laws

Kaplan et al. 2020 “Scaling Laws for Neural Language Models”

Larger models require **fewer samples** to reach the same performance.

The optimal model size grows smoothly with the loss target and compute budget.

- Test Loss
- Tokens Processed
- Compute (PF-days)

Line color indicates number of parameters:

- $10^3$
- $10^6$
- $10^9$

Compute-efficient training stops far short of convergence.
One-layer transformers learn “skip-gram” patterns similar to classical LMs. When a word occurs earlier in the document, the model learns to predict it again in an appropriate context later on.

**Primitive In-Context Learning Patterns**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
<th>Example 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[b]...[a] → [b]</code></td>
<td><code>[ two]...[ One] → [ two]</code></td>
<td><code>[ two]...[ has] → [ three]</code></td>
<td><code>[Ralph]...[ R] → [alph]</code></td>
<td><code>[Ralph]...[ R] → [ALPH]</code></td>
</tr>
<tr>
<td></td>
<td><code>[ perfect]...[ are] → [ perfect]</code></td>
<td><code>[ perfect]...[ looks] → [ super]</code></td>
<td><code>[Pike]...[ P] → [ike]</code></td>
<td><code>[Pike]...[ P] → [ikes]</code></td>
</tr>
<tr>
<td></td>
<td><code>[nbsp]...[ &amp;] → [nbsp]</code></td>
<td><code>[nbsp]...[ &amp;] → [gt]</code></td>
<td><code>[Pixmap]...[ P] → [ixmap]</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>[lambda]...[ $\backslash \backslash ] → [lambda]</code></td>
<td><code>[lambda]...[ $\backslash \backslash ] → [operator]</code></td>
<td><code>[ Lloyd]...[ L] → [loyd]</code></td>
<td></td>
</tr>
</tbody>
</table>
Multilayer transformers: “induction heads”

“Induction heads search over the context for previous examples of the present token… if they find it, they then look at the next token and copy it. This allows them to repeat previous sequences of tokens, both exactly and approximately.”
Linguistic abstractions

When Do You Need Billions of Words of Pretraining Data?
Zhang et al. 2020

Linguistic features are acquired rapidly between 1M and 10M words, but commonsense knowledge is much slower to acquire and probably requires billions
Sparsity

Although Transformers use mathematically dense representations, empirical work suggests that trained models are surprisingly sparse.

Li et al. “The Lazy Neuron Phenomenon: On Emergence of Activation Sparsity in Transformers” (2023)

Sparsity in variants of Google T5 (caveat: not a gigantic model!!)
Related phenomenon of “neural collapse”

In the final layer, embeddings of all data points within a label class converge to a single vector per class; these vectors form a simplex. (Caveat: vision model, not even remotely an LLM.) Kornblith et al. (2021) suggest such behavior might not be desirable for generalization.

Figure 1: Evolution of penultimate layer outputs of a VGG13 neural network when trained on the CIFAR10 dataset with 3 randomly selected classes. The green balls represent the coordinates of a simplex ETF, red ball-and-sticks represent the final layer classifier, blue ball-and-sticks represent the class means and the small blue balls represent the last hidden layer (penultimate layer) activations. Image credit: Papyan et al. (2020)
Discussion

The transformer architecture solves a variety of problems raised by earlier work:

- Layers of abstraction generalize across related words and context
- Non-Markov ability to reach far back in time
- Residual architecture provides clean gradients to every unit
- Massively over-parameterized model appears to allow effective learning

LLMs are mega-scale transformers, which enables them to learn faster per sample, and learn qualitatively new tricks (Winograd-style reasoning, instruction following…).
What I (at least) still don’t know

Transformers seem massively over-parameterized (capable of memorizing their training data), yet they still generalize effectively. What is (implicitly) regularizing them?

How do the dynamics of gradient descent interact with the model architecture to select the actual outcome of learning? How critical is the use of gradient optimization?

Loss curves for LLMs are generally fairly smooth, but there seem to be phase transitions in behavior and capability. What are the learning trajectories actually like? How do the learned representations reorganize themselves?

Massive scale appears critical to effective learning, but perhaps less so in deployment. How small can we compress LLMs? Can we use the same insights to train smaller ones in the first place?
Thank you for listening!