6.S079 LLMS IN A NUTSHELL

APRIL 2, 2024
MIKE CAFARELLA
Agenda

• Text Modeling
• Sequence Models
• Attention
• Transformers
• Fine-Tuning and RLHF
Agenda

• Text Modeling
• Sequence Models
• Attention
• Transformers
• Fine-Tuning and RLHF
The Core Thing

• We need a model that can compute $P(\text{someSentence})$

• We can use it to build a sequence model
  – Input: a sequence of tokens
  – Output: a sequence of tokens

• Lots of applications can be built this way
Autoregressive Text Generation

• If you can compute
  \[ P(\text{nextWord} \mid \text{precedingWords}) \]

• \[ P(\text{nextWord} \mid "My favorite food is") : \]
  – nextWord = “pizza”
  – nextWord = “love”
  – nextWord = “antagonist”

• Just try every possible word, pick the best
Autoregressive Text Generation

currentCtx = getUserInput()

while True:
    w = predictMostLikelyNextWord(currentCtx)
    emit(w)
    currentCtx += w

    if computeTerminationCriterion():
        break
Computing Sentence Probability

• For a sequence with n words
  \( w^n_1 = w_1, w_2, \ldots, w_n \)

• Every word \( w \) is drawn from a fixed vocabulary

• \( P(w^n_1) = P(w_1)P(w_2|w_1)P(w_3|w_2)\ldots P(w_n|w^{n-1}_1) \)

\[ = \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \]
Evaluation

• How can we tell if the model is good?
  – Maybe it helps with a downstream task
  – A general-purpose metric would be nice

• “Perplexity” measures the inverse probability of an unseen test corpus with a particular language model
  – If text is real, then its probability should be high
  – Lower is better

\[ PP(W) = P(w_1 \ldots w_N)^{-\frac{1}{N}} \]
Perplexity

- What’s nice about Perplexity?
  - It’s easy to compute
  - You don’t need a concrete task
  - You don’t need to understand the language!

Measures of Perplexity on different language models’ predictions of single-nucleotide sequences from prokaryotic genomes

From Nguyen et al, “Sequence modeling and design from molecular to genome scale with Evo”, 2024
Perplexity

• What’s bad about Perplexity?
  – It only works if your model gives a real probability (so: no rule-based methods)
  – Can’t compare language models with different vocabularies
  – What is a “good” Perplexity number?
Data, Models, Features

• More context is better
• …but we will run out of data for statistics when k-grams get big enough

• We need some combination of:
  – More informative features
  – Constrained models to avoid overfitting
• …but feature engineering for language is extremely hard
• …and expressivity of the model is hard to engineer
Neural Methods

• Neural methods like CBOW let us avoid feature engineering
• Managing overfitting is poorly-understood but works in practice through model architecture, dropout, and other methods
Encoder-Decoders

• Most sequence models use Encoder-Decoder architecture
  – The encoder converts the input into a compressed embedding-style representation
  – The decoder converts an encoded representation back into the target language

• Nice qualities:
  – You can train them separately
  – You can mix/match them for different input and output types

• word2vec has encoder/decoder architecture
Beyond w2v

• Remember CBOW? It aims to predict the output word given its context
• What’s bad about this for chat?
Weaknesses of w2v for chat

• Input architecture “looks into the future” (this is easy to fix)

• Each word has a single embedding, regardless of usage
  – “I am going to stick to it”
  – “I am going to throw the stick”
  – The w2v embedding for stick will reflect both senses, even though in some contexts the correct sense is obvious to a human

• Can’t handle truly huge vocabularies

• Sentence modeling is very primitive
Agenda

• Text Modeling
• Sequence Models
• Attention
• Transformers
• Fine-Tuning and RLHF
Sequence Modeling

• The input/output model of w2v is really basic
  – You put in some fixed-size context
  – You get out a word

• What if input/outputs are variable length?

• What if they have complicated structure?

Images thanks to Karpathy, “Unreasonable Effectiveness of Recurrent Neural Networks”
Sequence Modeling

• Sequence models offer more flexibility

- Red: input vectors, blue: output vectors
- Green: internal state
Sequence Modeling

one to many

many to one

many to many

many to many
Sequence Modeling

- **one to many**
- **many to one**
- **many to many**

Image captioning

Text labelling
Sequence Modeling

- one to many
- many to one
- many to many
- many to many
Sequence Modeling

one to many

many to one

many to many

Translation
Sequence Modeling

- one to many
- many to one
- many to many
- many to many
Sequence Modeling

(Hold off on chatbots for a bit...)
Sequence Modeling

- Internal state represents summary of inputs
- This breaks out of CBOB’s fixed-size context
- Simplest version is a Recurrent Neural Network
Detailed View of Char Model

- Task: predict the next character
- One input word is broken into four "labeled" pairs
- Input is one-hot encoded
- Output is softmax prediction vector
- $W_{xh}$ and $W_{hy}$ are init’ed randomly, then learned via training

Karpathy, “Unreasonable ...”
RNNs and seq2seq

- Sometimes we have an entire sequence to translate, summarize, or answer.
- We can train and combine two RNNs in an encoder/decoder “seq2seq” architecture.

Encoder RNN predicts next input. Decoder RNN takes encoder state and predicts next output.
RNNs and seq2seq

Allamar, Visualizing A Neural Machine Translation Model
RNN Problems

- Long-distance information passing isn’t great. First word of long seq may be “forgotten” by end
- In seq2seq architecture, the entire input sentence must be captured in one vector sent to decoder

Bahdanau, Cho, Bengio, “Neural Machine Translation By Jointly Learning to Align and Translate”
RNN Problems

Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL

Encoding Stage

Decoder RNN

Decoding Stage

Je
suis
étudiant

Allamar, Visualizing A Neural Machine Translation Model
RNN Problems

• Long-distance information passing isn’t great. First word of long seq may be “forgotten” by end.
• In seq2seq architecture, the entire input sentence must be captured in one vector sent to decoder (the “encoder bottleneck”).

• Various mechanisms invented to address this, such as Long Short-Term Memories.

• LSTMs had trainable components to intentionally “forget” parts of input and alleviate bottleneck.

• The “attention” mechanism was the most successful of these.
Agenda

• Text Modeling
• Sequence Models
• Attention
• Transformers
• Fine-Tuning and RLHF
Paying Attention

• Core idea: given some input, figure out the important parts and (mainly) ignore the rest
• How does your attention work when reading?

• The attention mechanism itself should be learned
• First introduced in “Neural Machine Translation By Jointly Learning to Align and Translate”, by Bahdanau, Cho, Bengio
• The decoder “chooses a subset of [encoder input] vectors adaptively while decoding the translation”
Bahdanau Attention

• Karpathy, AI’s bard: “The context vector from encoder is a weighted sum of hidden states of words of the encoding”

• Those attention weights are themselves computed by looking at current decoder state and encoder values

Bahdanau, Cho, Bengio, “Neural Machine Translation By Jointly Learning to Align and Translate”
Bahdanau Attention

Time step: 7

Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL WITH ATTENTION

Allamar, Visualizing A Neural Machine Translation Model
Bahdanau Attention

Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL WITH ATTENTION

Encoding Stage
Encoder RNN

Decoding Stage
Attention Decoder RNN

Je  suis  étudiant

Allamar, Visualizing A Neural Machine Translation Model
Attention Visualization

• Consider translating English to French
• Pixel values indicate relevance of input word to output word
Attention Visualization

- Consider translating English to French
- Pixel values indicate relevance of input word to output word
Attention Visualization

- Consider translating English to French
- Pixel values indicate relevance of input word to output word
Attention Visualization

- Consider translating English to French
- Pixel values indicate relevance of input word to output word
Attention Model

• Think about Queries, Keys, and Values
  • A Query describes what you’re trying to do ("find the predator")
  • A Key describes a particular input ("red things")
  • A Value describes the value of that input ("one big red thing near the lower right")
Attention Model

- Queries, Keys, and Values are vectors
Self-Attention

• Self-Attention works on an input sequence, allows us to drop the task-specific stuff
• We can encode both local and global dependencies
Self-Attention

As in the simplified version, this is a form of similarity or compatibility measure ("multiplicative attention").

For each query, model learns which key-value input it should attend to.

\[ A(q, K, V) = \sum_{i=1}^{T} \left[ \frac{\exp(q \cdot k_i^T)}{\sum_j \exp(q \cdot k_j^T)} \cdot v_i \right] \]

weighted sum: values weighted by attention weight (softmax score)

Intro to Deep Learning by Sebastian Raschka
Self-Attention

Input elements

As in the simplified version, this is a form of similarity or compatibility measure ("multiplicative attention")

For each query, model learns which key-value input it should attend to

\[ A(q_2, K, V) = \sum_{i=1}^{T} \left[ \frac{\exp(q_2 \cdot k_i^T)}{\sum_j \exp(q_2 \cdot k_j^T)} \times v_i \right] \]

weighted sum: values weighted by attention weight (softmax score)
Self-Attention

As in the simplified version, this is a form of similarity or compatibility measure ("multiplicative attention")

For each query, model learns which key-value input it should attend to

\[
A(q_2, K, V) = \sum_{i=1}^{T} \left[ \frac{\exp(q_2 \cdot k_i^T)}{\sum_j \exp(q_2 \cdot k_j^T)} \times v_i \right]
\]

weighted sum: values weighted by attention weight (softmax score)

Trainable weights

Intro to Deep Learning by Sebastian Raschka
Query, key, value vectors are result of combining input with learned weights

As in the simplified version, this is a form of similarity or compatibility measure ("multiplicative attention")

For each query, model learns which key-value input it should attend to

$$A(q, K, V) = \sum_{i=1}^{T} \left[ \frac{\exp(q_i k_i^T)}{\sum_j \exp(q_j k_j^T)} \times v_i \right]$$

weighted sum: values weighted by attention weight (softmax score)
We combine \( q_2 \) with each of the key \( k_i \) in order to obtain the attention we should pay to \( v_i \) and we do this for all queries.
Multi-Head Attention

- Compute the Self-Attention mechanism multiple times in parallel
- Using different sets of learned $W^q$, $W^k$, $W^v$ weights for each head
- More heads let us capture different kinds of attention
Multi-Head Attention

Allamar, The Illustrated Transformer
Karpathy’s View of Attention

• Consider a directed graph, where each node stores a vector.

• During “communication” nodes compute:
  • **Key vector**: What the node has
  • **Query vector**: What the node is looking for
  • **Value vector**: What the node will emit
Karpathy’s View of Attention

• Encoder attention creates “communication” or “message-passing” process
• Loop over the nodes randomly
• Each node combines its query vector with incoming nodes’ key vectors; yields “interestingness” score of each input
• Each node weights incoming values by score to yield an update to the node
• Attention in left-hand encoder “input” layer is fully connected, but masked MHA in decoder limits connectivity to reflect left-to-right text gen
Agenda

• Text Modeling
• Sequence Models
• Attention
• Transformers
• Fine-Tuning and RLHF
Transformers

- Most common architecture today is The Transformer
- “Attention is All You Need,” by Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin (2017)
- Drops recurrence; eats entire input
- Still autoregressive
Transformers

-most common architecture today is the Transformer

"Attention is All You Need," by Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin (2017)

-drops recurrence; eats entire input

Still autoregressive

https://www.researchgate.net/figure/The-Transformer-architecture-29-in-an-encoder-decoder-setting-adapted-to-facilitate_fig3_363085057
Transformers

MLP: Multilayer Perceptron, or, a fully-connected layer

https://www.researchgate.net/figure/The-Transformer-architecture-29-in-an-encoder-decoder-setting-adapted-to-facilitate_fig3_363085057
Transformers

- Multi-Head attention is computed in parallel
- Each “pane” is replicated in series N times
- Different weights in each setting
- Masked MHA disallows attention to “future” tokens
Transformers

INPUT: Je suis étudiant

OUTPUT: I am a student

Allamar, The Illustrated Transformer
Transformers

input: Je suis étudiant
output: I am a student

Allamar, The Illustrated Transformer
Agenda

- Text Modeling
- Sequence Models
- Attention
- Transformers
- Fine-Tuning and RLHF
Putting It All Together

• We train a big language model, then fine-tune

• FT retrains, using small dataset & limited weight updates

Ruder, https://www.ruder.io/recent-advances-lm-fine-tuning/
Fine Tuning

• Fine-tuning allows us to adapt to a particular domain or particular task
• Often enables real improvements with relatively small training sets

• **But**, what if we don’t have labeled data?
• **Also**, for a given output text, labeling it might be hard for a human, but choosing A vs B is easy
Reinforcement Learning with Human Feedback

• RLHF lets us use direct human feedback to fine-tune a model
• Often used to modify LLM tone, content guidelines, reducing “toxicity”

• Reinforcement Learning is AI area that uses “good dog/bad dog” signals instead of supervision
The RLHF Cycle

Prompt Dataset

"A dog is..."

Human-aligned LLM

"...man’s best friend"

RL algorithm

Reward Model

2.87

But how do we turn user preferences into a reward value?
The RLHF Cycle

**Prompts Dataset**

Sample many prompts

**Initial Language Model**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. AeneanDonec quam felis vulputate eget, arcu Nam quam nunc eros faucibus tincidunt, ultrices pulvinar, hendrerit.

**Human Scoring**

Generated text

**Reward (Preference) Model**

Outputs are ranked (relative, ELO, etc.)

Train on {sample, reward} pairs

The RLHF Cycle

1. Human beings rank lots of sample outputs

2. We turn each pairwise element in the ranking into a supervised label

The RLHF Cycle

3. Build a supervised model that takes an input text and predicts the users’ judgment.

4. This supervised model is the reward model for the RLHF cycle.
RLHF Has Its Problems

• How much RL is too much? Hard to figure out when to stop
• Training the reward model is one place we can make mistakes, then fine-tuning adds another
• **Direct Preference Optimization** (Rafailov et al) is a new scheme that avoids constructing the reward model; part of Lab 5!