NLP – The 100-Question Challenge

Ed Chen | edchen93@mit.edu

March, 2026

Abstract

This question-answer style handout served as exam review note for 6.8610 Quantitative Methods for Natural Language Processing at MIT. Ed took the course three times; the first version was written during the second, in Fall 2024, and revised during the third, in Spring 2026. These are Ed's personal notes from MIT 6.8610, not official course material. Assigned points reflect difficulty, from 3 to 9 per question and mostly 3 to 5. Where a question shows a sum such as 2+2+2, each part (a), (b), (c) carries its own points.

Contents

1 ML Intro 1
2 Embedding 4
3 Sequence Models 8
4 Attention 12
5 Transformer 15
6 Prompting 20
7 RLHF 23
8 Sampling 26
9 Efficiency 28
10 Retrieval and RAG 32
11 Agents and LM Programming 36
12 Special Topics 40

1 ML Intro

Question 1:

How do we quantify the accuracy of a logistic regression model's predictions during training, given that the model generates $\hat{y}$ predictions for m data points?
(3 points. Source: handout L01 p.103.)

Answer 1:

The performance of the model is measured using a cost function called "Cross-Entropy" or "Log Loss." The formula for the cost function is:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\Big[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})\Big] \tag{1}$$

This measures the difference between the predicted values $\hat{y}$ and the true labels y, where $\hat{y}$ is t′ predicted probability for the positive class and y is the actual label (1 or 0).

Question 2:

Compare the difference between forward-propagation, back-propagation, and gradient descent. About 2-4 sentences should be enough.
(3 points. Source: Ed modified from handout L01 p.107.)

Answer 2:

Forward-propagation calculates the predicted output of a neural network by passing inputs through the network's layers. Back-propagation computes the gradients of the loss function with respect to the network' s weights using the chain rule. Gradient descent in an algorithm that updates the weights using these gradients to minimize the loss function.

Question 3:

Explain data sparsity in the context of natural language processing and discuss its implications.
(3 points. Source: Ed modified from HW2.)

Answer 3:

Data sparsity refers to the phenomenon where certain word combinations occur infrequently or not at all in a dataset. For example, while the phrase "I love to read books" is common, "I love to eat books" might be rarely observed. This leads to challenges in accurately estimating probabilities for rare sequences, particularly as the length of these sequences increases.

Although infrequent, such phrases may still be crucial for understanding specific contexts or subtle meanings. Relying heavily on frequency metrics can cause models to overfit and introduce biases, potentially overlooking significant but rare data.

Question 4:

What does it indicate when validation loss is higher or lower than training loss during model training? (4 points. Source: Ed Chen.)

Answer 4:

Training loss is computed on the training dataset during model parameter updates, while validation loss is calculated on a separate validation set without updating the model.

When validation loss exceeds training loss, it typically indicates overfitting - the model has learned training data patterns too specifically and fails to generalize well to new data.

Conversely, when validation loss is lower than training loss, the cause is often benign: regularization such as dropout is active only during training, and the training loss is averaged over an epoch in which the model was still improving, while the validation loss is measured at the end of it. A validation set that happens to be easier than the training set has the same effect. A large or persistent gap can also point to data leakage, where validation data has inadvertently influenced the training process, commonly due to overlap between training and validation sets or preprocessing issues. That case requires careful investigation of the data pipeline or validation setup.

Question 5:

What is perplexity in language modeling? Give its formula, value range, and explain its meaning in 2-3 sentences. (3 points. Source: Ed modified from HW3.)

Answer 5:

Perplexity measures how well a language model predicts a text sequence, with lower values indicating better performance. It is calculated as the exponential of the average per-token negative log probability that the model assigns to the test sequence, essentially representing the weighted branching factor of the model's predictions.

$$\mathbf{PP} = \exp\left[-\frac{1}{N}\sum_{i=1}^{N}\log P(x_i)\right] \tag{2}$$

The value is at least 1 (perfect prediction) and has no upper bound. A model that is uniform over the vocabulary scores exactly the vocabulary size, and one that puts less than uniform probability on the actual tokens scores higher still. In practice, modern language models typically achieve perplexity between 15-60 on standard benchmarks.

Question 6:

For a three-class classifier, let z be the logits and y a one-hot target, with

$$p_i = \frac{\exp(z_i)}{\sum_k \exp(z_k)}, \qquad L = -\sum_i y_i \log p_i. \tag{3}$$

Use $z = (\ln 2, 0, 0)$ and $y = (0, 1, 0)$.

(a) Compute the probability vector and the loss.

(b) Derive $\partial L/\partial z_j$ for a general one-hot target, using the softmax derivative.

(c) Evaluate the three logit gradients and their sum.

(d) Take one gradient-descent step with learning rate 1 and give the new logits. Why must the gradient components always sum to zero?

(5 points. Source: Ed.)

Answer 6:

(a) The unnormalized probabilities are $(2, 1, 1)$ and sum to 4, so $p = (1/2, 1/4, 1/4)$. The target is class 2, giving $L = -\log(1/4) = \ln 4$.

(b) The softmax derivative is $\partial p_i/\partial z_j = p_i(\delta_{ij} - p_j)$. The chain rule gives $\partial L/\partial z_j = -\sum_i y_i(\delta_{ij} - p_j)$, and since $\sum_i y_i = 1$ this simplifies to $\partial L/\partial z_j = p_j - y_j$.

(c) The gradient is $(1/2, -3/4, 1/4)$, and its components sum to $1/2 - 3/4 + 1/4 = 0$.

(d) Subtracting the gradient gives $(\ln 2 - 1/2,\ 3/4,\ -1/4)$: the target logit rises and the other two fall. The components sum to zero because adding the same constant to every logit leaves the softmax, and so the loss, unchanged, which makes the derivative along $(1, 1, 1)$ zero.

Question 7:

Consider a two-layer scalar network with a ReLU hidden unit:

$$\begin{aligned} a &= w_1 x + b_1, \qquad & h &= \max(0, a), \\ \widehat{y} &= w_2 h + b_2, \qquad & L &= \tfrac{1}{2}(\widehat{y} - y)^2. \end{aligned} \tag{4}$$

Use $x = 2$, $y = 1$, and initial parameters $w_1 = 1$, $b_1 = -1$, $w_2 = 2$, $b_2 = 1$.

(a) Compute a, h, $\widehat{y}$ and L.

(b) Compute $\partial L/\partial \widehat{y}$, $\partial L/\partial w_2$ and $\partial L/\partial b_2$.

(c) Back-propagate to compute $\partial L/\partial h$, $\partial L/\partial a$, $\partial L/\partial w_1$ and $\partial L/\partial b_1$.

(d) Update all four parameters simultaneously by gradient descent with learning rate $1/4$, then recompute the prediction and the loss. What has happened to the hidden unit, and what does that mean for the next step's gradients to w_1 and b_1?

(5 points. Source: Ed.)

Answer 7:

(a) $a = 1(2) - 1 = 1$, so $h = 1$. Then $\widehat{y} = 2(1) + 1 = 3$ and $L = \frac{1}{2}(3 - 1)^2 = 2$.

(b) $\partial L/\partial \widehat{y} = \widehat{y} - y = 2$. Thus $\partial L/\partial w_2 = 2h = 2$ and $\partial L/\partial b_2 = 2$.

(c) $\partial L/\partial h = 2w_2 = 4$. Since $a = 1 > 0$, the ReLU derivative is 1, so $\partial L/\partial a = 4$. Therefore $\partial L/\partial w_1 = 4x = 8$ and $\partial L/\partial b_1 = 4$.

(d) The updated parameters are $w_1 = 1 - 8/4 = -1$, $b_1 = -1 - 4/4 = -2$, $w_2 = 2 - 2/4 = 3/2$ and $b_2 = 1 - 2/4 = 1/2$. The new pre-activation is $(-1)2 - 2 = -4$, so $h = 0$, $\widehat{y} = 1/2$ and $L = \frac{1}{2}(1/2 - 1)^2 = 1/8$. The loss fell, but the hidden unit is now dead on this input: with $a < 0$ the ReLU derivative is 0, so the next step sends no gradient to w_1 or b_1, and only b_2 keeps learning.

Question 8:

Adam (Kingma and Ba, 2014, arXiv:1412.6980) initializes its moment estimates at zero and applies bias correction:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t, \quad & \widehat{m}_t &= \frac{m_t}{1 - \beta_1^t}, \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \quad & \widehat{v}_t &= \frac{v_t}{1 - \beta_2^t}. \end{aligned} \tag{5}$$

For one scalar parameter, use $m_0 = v_0 = 0$, $\beta_1 = 1/2$, $\beta_2 = 15/16$, $g_1 = 4$, $\theta_0 = 3$, learning rate $\alpha = 1/4$ and $\epsilon = 0$. The update subtracts $\alpha \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$.

(a) Compute m_1 and v_1.

(b) Compute the corrected moments and the updated parameter θ_1.

(c) Suppose both corrections are omitted. Compute the resulting parameter and compare the step size with (b).

(d) Assume every gradient has the same mean μ and second raw moment ν. Give $\mathbb{E}[m_t]$ and $\mathbb{E}[v_t]$ under zero initialization, and say exactly what bias the correction removes.

(5 points. Source: Ed.)

Answer 8:

(a) $m_1 = (1/2)4 = 2$ and $v_1 = (1/16)4^2 = 1$.

(b) $\widehat{m}_1 = 2/(1/2) = 4$ and $\widehat{v}_1 = 1/(1/16) = 16$. The step is $(1/4) \times 4/\sqrt{16} = 1/4$, so $\theta_1 = 3 - 1/4 = 11/4$. At $t = 1$ the corrected step is always exactly α in the direction of $-g_1$, whatever the size of g_1.

(c) The step is $(1/4) \times 2/\sqrt{1} = 1/2$, giving $\theta_1 = 5/2$, twice the corrected step. The direction of the error depends on the decay rates: the uncorrected second moment is shrunk more than the first here, because β_2 is closer to 1, so the division overshoots.

(d) Unrolling the recurrences gives $\mathbb{E}[m_t] = (1 - \beta_1^t)\mu$ and $\mathbb{E}[v_t] = (1 - \beta_2^t)\nu$. Zero initialization pulls both estimates toward zero in the early steps, and dividing by $1 - \beta^t$ removes exactly that factor. The bias is from the initialization, not from noise in the gradients, and it fades as $\beta^t \to 0$.

2 Embedding

Question 9:

Consider the term-document matrix (term frequency $\text{TF}_{(t,d)}$) shown in Equation (6), where each entry $\text{TF}_{(a,b)}$ represents the number of times word a appears in document b:

$$W_{td} = \begin{array}{c} \\ \text{cat} \\ \text{dog} \\ \text{the} \end{array} \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & d_3 & d_4 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 2 & 0 & 1 \\ 20 & 13 & 18 & 22 \end{bmatrix} \end{array} \tag{6}$$

Write the equation for Inverse Document Frequency (IDF) and explain how TF-IDF addresses the issue where common but less meaningful words like "the" have high frequency counts across documents.
(5 points. Source: Ed modified from HW1.)

Answer 9:

The IDF for term i is given by:

$$\text{IDF}_i = \log\left(\frac{N}{\sum_{j=1}^{N} \mathbf{1}(t_i \in d_j)}\right) \tag{7}$$

where N is the total number of documents, and $\mathbf{1}(t_i \in d_j)$ equals 1 if term i appears in document j, and 0 otherwise.

TF-IDF addresses the frequency bias by multiplying each term frequency by its IDF value. Since common words like "the" appear in most documents, they receive a low IDF score (close to 0), reducing their overall impact. Conversely, distinctive words appearing in fewer documents receive higher IDF scores, making them more significant in the final representation.

Question 10:

What are the key differences between TF-IDF and neural word embeddings? List at least two distinctions.
(4 points. Source: Midterm practice bigram-w-LHS.)

Answer 10:

- Representation scope: TF-IDF represents entire documents or paragraphs, while word embeddings represent individual words or subword tokens.
- Dimensionality: TF-IDF creates sparse, high-dimensional vectors based on vocabulary size, while neural embeddings produce dense, lower-dimensional vectors that capture semantic relationships.
- Training approach: TF-IDF is a statistical method based on word frequency counts, while neural embeddings are learned through training on large text corpora to capture word meanings and relationships.

Question 11:

For deep learning models, we have an embedding layer and hidden layer(s). The hidden layers tend to use non-linear activation functions. Why do we desire such, and why not for the embedding layer? Answer with a few sentences is fine.
(3 points. Source: Midterm practice let-w-unigram.)

Answer 11:

The embedding layer allows us to quickly represent each input word with a distinct embedding. Any contextual relations are subsequently learned; but, we must first allow words to be represented as neural embeddings. Initializing by trying to capture non-linear relationships is too unnecessary, overkill, and wouldn't make sense.

Question 12:

Consider two identical Skip-Gram word2vec models trained the same way, except one used reversed input text sequences. How would the learned embeddings compare between these models, and why? (2-3 sentences)
(3 points. Source: Midterm practice bigram-w-LHS.)

Answer 12:

The embeddings from both models would be identical because Skip-Gram learns word representations by predicting context words within a fixed window, regardless of their order. Since reversing the text preserves which words appear together within each window, both models encounter the same word-context pairs during training, resulting in equivalent embeddings.

Question 13:

Answer the questions based on the model architecture in Figure 1.

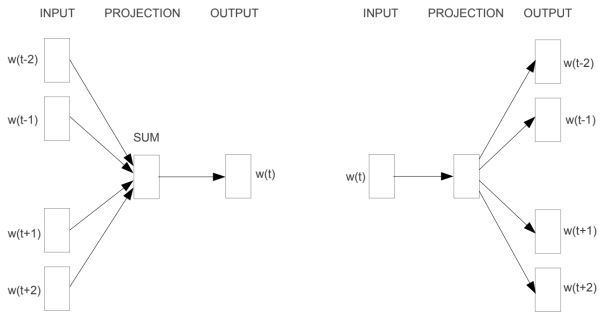


Figure 1: Model Architecture

(a) Identify the components and the overall technique used in this architecture.

(b) Is this a language model or a technique? Explain.

(c) The following equation is related to this technique:

$$w_x \leftarrow w_x + \eta \left(u_y - \mathbb{E}_{p_\theta(v|x)}\left[u_v\right]\right) \tag{8}$$

Explain the meaning of each main terms in the equation.

Answer 13:

(a) This architecture represents Word2Vec (2013, arXiv:1301.3781), a method to convert words into vector representations based on their context.
The left part of the diagram is the Continuous Bag of Words (CBOW), which predicts the target word based on its surrounding context. The right part is Skip-gram, which predicts the surrounding context given a target word.

(b) Word2Vec is not a language model; it is a technique used to generate word embeddings. It learns vector representations of words based on their contextual relationships within a corpus using a neural network.

(c) The equation describes the weight update for word embeddings:

- u_y: The embedding for the current word (label).
- $\mathbb{E}_{p_\theta(v|x)}\left[u_v\right]$: The expected embedding over possible words generated from context x.
- η: The learning rate, which controls the step size in the weight update.
- w_x: The embedding of the current input word x, updated based on the difference between the actual and expected embeddings.

Question 14:

A language model represents a distribution $p\left(x_i \mid \mathbf{x}_{:i-1}\right)$ over next tokens x_i given contexts $\mathbf{x}_{:i-1}$. Suppose we know we want to generate n tokens from an LM. How do we find a high-probability sequence $\mathbf{x}$ without enumerating and scoring all possible length- n sequences?

One simple, approximate approach we've seen in class is greedy search. In greedy search, we always generate the next token after a prefix x by choosing:

$$\text{Greedy}_1(\mathbf{x}) = \arg\max_{\mathbf{x}'} p\left(x' \mid \mathbf{x}\right)$$

Let's denote the result of running greedy search for k steps on an input $\mathbf{x}$ as follows:

$$\begin{aligned} \mathbf{x}' &= \text{Greedy}_{k-1}(\mathbf{x}) \\ \text{Greedy}_k(\mathbf{x}) &= \text{Greedy}_1\left(\mathbf{x}'\right) \oplus \mathbf{x}' \qquad (\text{for } k > 1) \end{aligned}$$

where $\oplus$ denotes string concatenation. (This is fancy notation for a simple definition: to run greedy search for k steps, we choose the first token greedily, then add it to the context and repeat for k steps.)

(a) Write down three strings, each consisting of a start symbol followed by two tokens, and assign each string a probability, so that greedy decoding is suboptimal: $\text{Greedy}_3(\langle s \rangle) \neq \arg_{\text{g}} \max_{\text{x}} p(\mathbf{x})$.

(b) Now let's consider an alternative decoding procedure, which we'll call search with lookahead. In this procedure, we pick the token that would be the best starting point for a k-step greedy search:

$$\text{Lookahead}_1^k(\mathbf{x}) = \arg\max_{x'} p\left(\text{Greedy}_k\left(\mathbf{x} \oplus x'\right)\right)$$

(where, as above, $\oplus$ denotes string concatenation). If our vocabulary size is n, how many times do we need to query the LM$p(x)$ to generate a single token using Lookahead $_1^k$? (Big $-$ O is fine.)

(c) How many times do we need to query the LM to generate m tokens? (Can we re-use any computation?)

(d) Write down a distribution of four strings, as above, where Greedy 3 and Lookahead $_1^3$ produce different answers, and neither of them produces the most probable string. These strings may be of any length.

Answer 14:

(a) To demonstrate suboptimal greedy decoding, consider these three strings with associated probabilities:

$$\begin{aligned} \langle s \rangle \text{ a a} &: 0.26 \\ \langle s \rangle \text{ a b} &: 0.26 \\ \langle s \rangle \text{ b b} &: 0.48 \end{aligned}$$

Greedy decoding will choose 'a' as the first token, leading to either "aa" or "ab", both suboptimal compared to the most probable string "bb".

(b) For Lookahead$_1^k$ with vocabulary size n, we need to:

- Consider n possible first tokens

- For each, perform k steps of greedy search
- Each greedy step requires n LM queries

Total LM queries: $O(n^2k)$

(c) To generate m tokens:

- Repeat the process m times
- Reuse one sequence of greedily-chosen tokens from the previous step

Total LM queries: $O(m(n-1)nk)$, that is $O(mn^2k)$

(d) Consider these four strings with associated probabilities:

$$\langle s\rangle \text{ abbb} : 0.30$$
$$\langle s\rangle \text{ accc} : 0.23$$
$$\langle s\rangle \text{ accd} : 0.22$$
$$\langle s\rangle \text{ bbbb} : 0.25$$

Greedy picks "a" (0.75 against 0.25), then "c" (0.45 against 0.30), then "c" and "c", ending at "accc" (0.23). Lookahead$_1^3$ compares the greedy continuation of each first token, "accc" (0.23) against "bbbb" (0.25), so it picks "b" and ends at "bbbb". Neither produces the most probable string "abbb" (0.30).

Question 15:

Use the following two-dimensional word vectors: $e_{\text{king}} = (4,1)$, $e_{\text{man}} = (1,0)$, $e_{\text{woman}} = (0,3)$, $e_{\text{queen}} = (3,4)$, $e_{\text{duchess}} = (0,5)$ and $e_{\text{prince}} = (5,0)$. For nonzero vectors,

$$\cos(u,v) = \frac{u^\top v}{\|u\|_2\|v\|_2}. \tag{9}$$

For analogy retrieval, search only among queen, duchess and prince, leaving out the three query words as is standard.

(a) Compute the norms of the three candidates and the cosine similarity of queen with duchess and with prince.

(b) Form the analogy query $q = e_{\text{king}} - e_{\text{man}} + e_{\text{woman}}$ and compute its norm.

(c) Compute the cosine similarity of q with each candidate and select the answer.

(d) Replace only the queen vector by $(6,8)$. Recompute its cosine similarity with q, and give the Euclidean distance from q to the original and to the replacement queen vector.

(4 points. Source: Ed.)

Answer 15:

(a) All three norms are 5; for queen, $\sqrt{3^2+4^2} = 5$. Queen with duchess has dot product 20 and cosine $20/25 = 4/5$; queen with prince has 15 and cosine $3/5$.

(b) $q = (4-1+0,\ 1-0+3) = (3,4)$, with norm 5.

(c) Queen $25/25 = 1$, duchess $20/25 = 4/5$, prince $15/25 = 3/5$. The answer is queen.

(d) The new vector has norm 10 and dot product $3(6)+4(8) = 50$ with q, so its cosine is $50/(5\times 10) = 1$, unchanged. The Euclidean distance goes from 0 to $\sqrt{3^2+4^2} = 5$. Cosine similarity ignores length and Euclidean distance does not, which is why embedding search normally uses cosine: frequent words tend to have longer vectors without being closer in meaning.

Question 16:

fastText builds word representations from character n-grams (Bojanowski et al., 2016, arXiv:1607.04606). Consider a toy version using only character trigrams. Add the boundary symbols < and > before extracting all contiguous length-three substrings, and assume no hashing collisions. The representation is

$$e(w) = u_w + \sum_{g\in G(w)} z_g, \tag{10}$$

where u_w is a separate whole-word vector, zero for an unseen word. Let $u_{\text{cat}} = (1,1)$, $z_{\texttt{<ca}} = (1,0)$, $z_{\texttt{cat}} = (0,2)$, $z_{\texttt{at>}} = (1,1)$, and let $z_{\texttt{<ba}} = z_{\texttt{bat}} = (0,0)$, never having been trained. The word `bat` is unseen.

(a) Enumerate the trigrams of `cat` with their count. Is the whole-word vector one of these trigrams?

(b) Compute $e(\text{cat})$.

(c) Enumerate the trigrams of `bat` and compute its representation.

(d) Name the shared trigram that gives `bat` its nonzero representation. What would word2vec give for `bat`, and why does that make subword models attractive for morphologically rich languages?

(4 points. Source: Ed.)

Answer 16:

(a) The marked string `<cat>` gives three trigrams, `<ca`, `cat` and `at>`. The whole-word vector is not one of them. It is a separate parameter for the full sequence `<cat>`, even though one trigram happens to spell `cat`.

(b) $e(\text{cat}) = (1,1) + (1,0) + (0,2) + (1,1) = (3,4)$.

(c) `<bat>` gives `<ba`, `bat` and `at>`. With no whole-word vector, $e(\text{bat}) = (0,0) + (0,0) + (1,1) = (1,1)$.

(d) The shared trigram is `at>`. Word2vec has one vector per vocabulary word, so an unseen word has no representation at all and must be mapped to an unknown token. Morphologically rich languages generate many rare inflected forms of each stem, and a subword model can compose a sensible vector for each from the stem and affix pieces it has already learned.

Question 17:

Skip-gram with negative sampling (Mikolov et al., 2013, arXiv:1310.4546) raises the score of a center word v_c with an observed context word u_o and lowers it with k sampled noise words u_{n_i}:

$$\mathcal{L} = -\log \sigma(v_c^\top u_o) - \sum_{i=1}^{k} \log \sigma(-v_c^\top u_{n_i}). \tag{11}$$

(a) Noise words are drawn with probability proportional to $C(w)^{3/4}$, where $C(w)$ is the unigram count. For three words with counts 1, 16 and 81, compute the noise distribution, and compare it with sampling proportional to the raw counts.

(b) Why is this loss cheaper per update than a full softmax over the vocabulary?

(c) Take $k = 1$, $v_c^\top u_o = \ln 3$ and $v_c^\top u_n = -\ln 3$. Compute $\mathcal{L}$.

(d) For the same values, compute $\partial \mathcal{L} / \partial v_c$ as a combination of u_o and u_n.

(5 points. Source: Ed.)

Answer 17:

(a) $1^{3/4} = 1$, $16^{3/4} = 2^3 = 8$ and $81^{3/4} = 3^3 = 27$, which sum to 36, so the probabilities are $1/36$, $2/9$ and $3/4$. Raw counts would give $1/98$, $16/98$ and $81/98$. The exponent moves mass from the most frequent word to the rare ones, so rare words are drawn as negatives more often than their frequency alone would allow.

(b) A full softmax needs a dot product with every word in the vocabulary to normalize, which can be millions per update. Negative sampling replaces that with $k + 1$ binary classifications, so each update costs $O(k)$.

(c) The positive term is $-\log \sigma(\ln 3) = \ln(4/3)$. The negative term is $-\log \sigma(\ln 3)$ as well, since $-v_c^\top u_n = \ln 3$. The total is $2\ln(4/3)$.

(d) The gradient is $-(1 - \sigma(v_c^\top u_o))\, u_o + \sigma(v_c^\top u_n)\, u_n$. With $1 - \sigma(\ln 3) = 1/4$ and $\sigma(-\ln 3) = 1/4$, this is $-\frac{1}{4} u_o + \frac{1}{4} u_n$. A descent step moves v_c toward the context word and away from the noise word.

3 Sequence Models

RNN, GRU, LSTM. (Transformer is in 5).

Question 18:

Consider a simple Recurrent Neural Network (RNN) defined by the following equations:

$$h_t = f\left(W_h h_{t-1} + W_x x_t\right)$$
$$y_t = W_y h_t$$

Where: - $h_t \in \mathbb{R}^2$ is the hidden state at time step t - $x_t \in \mathbb{R}$ is the scalar input at time step t - $W_h \in \mathbb{R}^{2\times 2}, W_x \in \mathbb{R}^{2\times 1}, W_y \in \mathbb{R}^{1\times 2}$ - There are no bias terms - The activation function is the identity $f(z) = z$

You are given the initial hidden state $h_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and the parameters $W_h = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, W_y = \begin{bmatrix} 1 & -1 \end{bmatrix}$.

(a) (3 points) Is there a particular weight matrix W_x such that the RNN produces the output sequence $y_1 = 0.5, y_2 = 0.2$ when fed the input sequence $x_1 = 1, x_2 = 1$? If so, derive the matrix W_x that would result in the desired outputs. If not, prove that this is impossible.

(b) (3 points) Suppose we're now using a ReLU (applied element-wise) as the activation function. Is it possible for the model to produce the desired outputs? If so, derive the matrix W_x that would result in the desired outputs. If not, prove that this is impossible.
Note: Recall that $\text{ReLU}(z) = \max(0, z)$.

(6 points. Source: Fall 2024, MIT NLP midterm.)

Answer 18:

(a) The answer is no. Let $W_x = \begin{bmatrix} a \\ b \end{bmatrix}$. Given $x_1 = 1$, at $t = 1$, we have:

$$h_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} 1 = \begin{bmatrix} a \\ b \end{bmatrix}$$
$$y_1 = W_y h_t = \begin{bmatrix} 1 & -1 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = a - b \tag{12}$$

Follow the given equation, at $t = 2$, given $x_2 = 1$, we arrive:

$$h_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} 1 = \begin{bmatrix} a+b \\ a+b \end{bmatrix}$$
$$y_2 = \begin{bmatrix} 1 & -1 \end{bmatrix}\begin{bmatrix} a+b \\ a+b \end{bmatrix} = 0 \tag{13}$$

As we can tell, any $W_x = \begin{bmatrix} a \\ b \end{bmatrix}$ will result in $y_2 = 0$.

(b) Solution: The answer is yes. Suppose $W_x = \begin{bmatrix} a \\ b \end{bmatrix}$. At $t = 1$, given $x_1 = 1$ and $y_1 = 0.5$ we have

$$h_1 = \text{ReLU}\left(\begin{bmatrix} a \\ b \end{bmatrix}\right) = \begin{bmatrix} \max(0, a) \\ \max(0, b) \end{bmatrix}$$
$$y_1 = W_y h_1 = \begin{bmatrix} 1 & -1 \end{bmatrix} h_t \tag{14}$$
$$= \max(0, a) - \max(0, b) := 0.5$$

At $t = 2$, given $x_2 = 1$, we arrive:

$$h_2 = \text{ReLU}\left(\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} h_1 + \begin{bmatrix} a \\ b \end{bmatrix} 1\right)$$
$$= \text{ReLU}\left(\begin{bmatrix} \max(0, b) + a \\ \max(0, a) + b \end{bmatrix}\right)$$
$$= \begin{bmatrix} \max(0, \max(0, b) + a) \\ \max(0, \max(0, a) + b) \end{bmatrix} \tag{15}$$
$$y_2 = W_y h_2 = \begin{bmatrix} 1 & -1 \end{bmatrix} h_2 := 0.2$$

We arrived:

$$\begin{aligned} \rightarrow \quad & \max(0, \max(0, b) + a) \\ & - \max(0, \max(0, a) + b) = 0.2 \end{aligned} \tag{16}$$

Solving possible combination of W_x.

Since $\max(0, a) - \max(0, b) = 0.5 > 0$, we need $\max(0, a) > 0$, so $a > 0$ and $\max(0, a) = a$.

Case 1: $b \geq 0$**.** Then $\max(0, b) = b$ and the y_1 constraint gives $a - b = 0.5$. Substituting into the y_2 constraint, both components of h_2 become $\max(0,\ a + b)$:

$$\max(0,\ b + a) - \max(0,\ a + b) = 0 \neq 0.2, \tag{17}$$

so Case 1 is impossible.

Case 2: $b < 0$**.** Then $\max(0, b) = 0$ and $a = 0.5$. The y_2 constraint becomes:

$$\begin{aligned} &\max(0,\ 0.5) - \max(0,\ 0.5 + b) \\ =&0.5 - \max(0,\ 0.5 + b) \\ =&0.2 \end{aligned} \tag{18}$$

so $\max(0,\ 0.5 + b) = 0.3$, implying $0.5 + b = 0.3$, hence $b = -0.2$. This is consistent: $b = -0.2 < 0$ and $0.5 + b = 0.3 > 0$. ✓

Therefore $W_x = \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}$.

Question 19:

Which of the following models are sequence models? Give short reason to the models that are not sequence models.
Models: Feed-Forward Neural Networks, Decision Trees, Bag-of-Words (BoW), Latent Dirichlet allocation (LDA), Word2Vec, N-gram Models, Hidden Markov Models (HMMs), Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRU), Long Short-Term Memory (LSTM), Transformers, 1D-CNN.
(5 points. Source: Ed Chen.)

Answer 19:

Sequence models: N-gram Models, Hidden Markov Models (HMMs), Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRU), Long Short-Term Memory (LSTM), Transformers, 1D-CNN.

Not sequence models:

- Feed-Forward Neural Networks: No inherent sequential processing; treats inputs independently.
- Decision Trees: Treat features independently.
- Bag-of-Words (BoW): Ignores word order completely.
- Latent Dirichlet allocation (LDA): Focus on discovering topics, not sequence patterns.
- Word2Vec: It's a process to create static word representations

Question 20:

It is often said that RNN < GRU < LSTM < Transformer in terms of performance. Please answer the following questions using one sentence each.

(a) What are the limitations of RNN?

(b) How does GRU improve upon RNN?

(c) How does LSTM improve upon GRU?

(d) How does Transformer improve upon LSTM?

(4 points. Source: Ed Chen.)

Answer 20:

(a) RNN struggles with learning long-term dependencies due to vanishing gradients, making it less effective for sequential tasks over longer durations.

(b) GRU introduces update and reset gates, which help capture longer-term dependencies and mitigate the vanishing gradient problem with fewer parameters than LSTM.

(c) LSTM, which predates GRU, keeps a separate cell state controlled by forget, input, and output gates, giving finer control over what is stored and what is exposed at the cost of more parameters; empirical comparisons generally find the two comparable rather than LSTM consistently ahead.

(d) Transformer uses self-attention mechanisms, eliminating the need for sequential data processing and enabling it to handle longer dependencies and large datasets more efficiently.

Question 21:

Given the conditional probability of a word pair in a bigram model:

$$p\left(w_i \mid w_{i-1}\right) = \frac{\text{Count}\left(w_{i-1}, w_i\right)}{\text{Count}\left(w_{i-1}\right)} \tag{19}$$

Write the joint probability of a sequence $x_1, x_2, \ldots, x_T$ using a capital Greek letter Pi.
(3 points. Source: Ed Chen.)

Answer 21:

$$P_{\text{bigram}}(x_1, x_2, \ldots, x_T) = p(x_1) \prod_{t=2}^{T} p(x_t \mid x_{t-1}) \tag{20}$$

Question 22:

Explain the concept represented by this equation:

$$\mathrm{P}\left(w_t \mid w_1, \ldots, w_{t-1}\right) \approx \mathrm{P}\left(w_t \mid w_{t-n}, \ldots, w_{t-1}\right) \tag{21}$$

(3 points. Source: Ed Chen.)

Answer 22:
This equation represents the Markov assumption in language modeling, which states that the probability of a word w_t can be approximated by considering only the previous n words, rather than the entire history. Conditioning on n previous words makes this an $(n+1)$-gram model. This simplification makes language modeling computationally tractable, though it limits the model's ability to capture long-range dependencies.

Question 23:
How does a basic RNN architecture work? Draw a diagram showing its layers and connections.
(4 points. Source: Ed Chen.)

Answer 23:

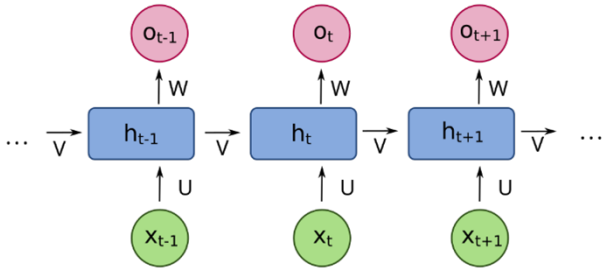


Figure 2: Diagram of basic recurrent neural network.

A Recurrent Neural Network (RNN) processes sequential data through recurrent connections, where each hidden state h_t depends on both the current input x_t and previous hidden state h_{t-1}.

- Input layer (green nodes x_t) feeding into hidden states via weight matrix U.
- Hidden states (blue boxes h_t) connected to themselves via weight matrix V.
- Output layer (pink nodes o_t) generated from hidden states via weight matrix W.

The recurrent connection V between hidden states allows the network to maintain information across time steps.

Question 24:
Answer the following questions regrading RNN with one sentence each.

(a) What can RNNs capture that Word2Vec cannot?

(b) What can RNNs predict that Word2Vec cannot?

(c) How does CBOW differ from an RNN?

(d) What are the main computational challenges in training RNNs?

(4 points. Source: Ed, modified from handout L14.)

Answer 24:

(a) RNNs can represent sequential patterns and temporal dependencies that Word2Vec cannot, as it only models local context within a fixed window.

(b) RNNs can predict the next token in a sequence by accounting for the entire sequence history, whereas Word2Vec focuses on predicting words based on fixed context windows.

(c) CBOW aggregates context into a single prediction without considering order, while RNNs process sequences in an ordered manner, retaining past states through recurrent connections.

(d) Training is slow because back-propagation through time must process the sequence step by step and cannot be parallelized across time steps, and unstable because gradients vanish or explode as they pass back through many steps.

Question 25:
Recall that there exists a part-of-speech for each word in a sentence, for example,

Juyeon's	father	sleeps
Poss-Noun	Noun	Verb

, where Poss-Noun stands for Possessive Noun. For this question, let us consider perplexity of words aggregated by part-of-speech tags, where the perplexity aggregated by part-of-speech tag k (where $k \in \{$ Noun, Verb, ... $\}$) is computed by the exponential of the average negative log likelihood of words with the tag k. Call this quantity perplexity (k). English verbs in general do not have an honorific form. However, verbs in some languages, such as Korean, do. For example, the following formal expression:

Juyeon-ui	abeoji-neun	jumushinda
Juyeon's	father	sleeps $_{\text{formal}}$

should be contrasted with the informal form of "sleeps",

Juyeon-ui	abeoji-neun	janda
Juyeon's	father	sleeps $_{\text{informal}}$

The use of the informal form in the above sentence is grammatical, but would be considered unusual and would certainly raise some eyebrows. (The deliberate use of an informal form in social contexts in which one is expected to use the formal form is usually done so to make a point, e.g., to show that you do not respect Juyeon or Juyeon's father.)

Now consider a version of English, called BTS-English, which expands (some) verbs to have both an informal and a formal form. Thus, the set of verbs is now expanded to be to be larger, e.g.,

$$\{\ \text{ate}_{\text{formal}},\ \text{ate}_{\text{in formal}},\ \text{talk}_{\text{formal}},\ \text{talk}_{\text{informal}},\ldots\}$$

BTS-English is like Korean in that one is expected to use an honorific form of the verb in cases where the entity being addressed should be shown respect. Please answer the following questions, with a brief explanation (1-2 sentences).

(a) (2 points) Suppose one uses a unigram language model. Would perplexity(Noun) in BTS-English be higher, lower, or the same compared to English?

(b) (2 points) Suppose one uses a uniform language model where the probability a word is uniformly the same. Would perplexity (Verb) in BTS-English be higher, lower, or the same compared to English?

(c) (2 points) Suppose one uses a trigram language model. Would perplexity(Verb) in BTS-English be higher, lower, or the same compared to English? (Hint: English is an SVO language where subject nouns generally come before verbs).

(6 points. Source: Midterm practice bigram-w-LHS.)

Answer 25:

(a) Unchanged, since the number of nouns is unchanged.

(b) Higher, as there are more verbs.

(c) Two answers are possible:
Lower, since conditioning on subject nouns reduces uncertainty.
Higher; while conditioning on subject nouns reduces uncertainty, just conditioning on the previous two tokens is not "enough", and the reduction in uncertainty is offset by the increase in uncertainty due to there being more verbs.

Question 26:

An LSTM protects its cell state with gates. The cell and hidden updates are

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t, \qquad h_t = o_t \tanh(C_t). \tag{22}$$

Let $C_{t-1} = 4$, and suppose step t produces a forget gate $f_t = 1$, an input gate $i_t = 1/2$, an output gate $o_t = 0$ and a candidate $\tilde{C}_t = 2$.

(a) Compute the new cell state C_t.

(b) Compute the new hidden state h_t.

(c) Describe in words what the cell did with its history, with the new input, and with its output at this step.

(d) Give $\partial C_t / \partial C_{t-1}$ along the direct cell path. If ten consecutive steps all have $f = 1$, what is $\partial C_{10} / \partial C_0$ along that path? Compare with a recurrence whose per-step factor is $1/2$, and explain why the additive update eases vanishing gradients.

(5 points. Source: Ed.)

Answer 26:

(a) $C_t = 1 \times 4 + \frac{1}{2} \times 2 = 5$.

(b) $h_t = 0 \times \tanh(5) = 0$.

(c) With $f_t = 1$ the cell kept its whole history; with $i_t = 1/2$ it wrote half of the candidate on top; with $o_t = 0$ it showed none of this to the rest of the network. The information is stored but hidden, and a later step can reveal it by opening the output gate.

(d) Along the direct path $\partial C_t / \partial C_{t-1} = f_t = 1$, so ten such steps give $1^{10} = 1$. A recurrence that multiplies by $1/2$ each step gives $(1/2)^{10} = 1/1024$. In a plain RNN the factor is a weight matrix times a squashing derivative, applied again at every step, so it tends to shrink geometrically. The LSTM cell is updated by addition, and its factor is the forget gate itself, which the network can learn to hold near 1 for as long as the information is needed.

4 Attention

Question 27:

Self-Attention is powerful. It allows us to create great, context-aware representations.

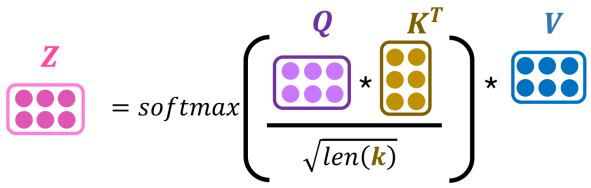


Figure 3: Self-attention.

(a) What is the role of the Query, Key, and Value vectors in the self-attention mechanism, and how are attention scores computed? (1 - 2 sentences for each of the would be sufficient.)

(b) Why do we divided by square root of the length of Key, rather than just the length of it? (1 - 3 sentence of explanation.)

(c) Can all Z be calculated at the same time, via matrix multiplications? (1 - 3 sentence of explanation.)

(9 points. Source: Ed Chen.)

Answer 27:

(a) In the self-attention mechanism, the Query (Q), Key (K), and Value (V) vectors are linear projections of the input embeddings:

- Query (Q): represents the token that is "asking" for relevant information.
- Key (K): represents how each token can be matched against queries.
- Value (V): represents the content that will be aggregated.

Attention scores are $QK^\top/\sqrt{d_k}$, and the output is

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \tag{23}$$

(b) The dot product $q^\top k$ grows in magnitude with dimension d_k, increasing its variance. Without scaling, large values push softmax into saturation and harm gradients. Dividing by $\sqrt{d_k}$ keeps the variance roughly constant and stabilizes training. If the components of q and k have unit variance, $q^\top k$ has variance d_k and standard deviation $\sqrt{d_k}$, so $\sqrt{d_k}$ is exactly the factor that restores unit variance. Dividing by d_k would shrink the scores toward zero as d_k grows and flatten the softmax toward uniform.

(c) Yes. All outputs Z are computed in parallel via matrix multiplications:

$$Z = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V.$$

The softmax is applied row-wise, enabling full parallelization across tokens.

Question 28:

Explain the following three types of attention mechanisms in 1–2 sentences each.

(a) Self-Attention

(b) Masked Self-Attention

(c) Cross-Attention

Answer 28:

(a) **Self-Attention:** Computes attention within a single sequence by scoring each token against all other tokens in that sequence, producing context-aware representations.

(b) **Masked Self-Attention:** Same as self-attention but prevents tokens from attending to future positions (e.g., in auto-regressive decoding) by masking out those logits before softmax.

(c) **Cross-Attention:** Computes attention from one sequence over another (e.g., queries from the decoder attend to keys/values from the encoder) to incorporate information from a different source.

Question 29:

Consider a sequence-to-sequence (seq2seq) model built from a recurrent encoder and a recurrent decoder, first without attention and then with it.

(a) What does it mean for a model to perform conditional prediction? Provide an example. (1-2 sentences)

(b) What is the purpose of using a separate encoder and decoder architecture? (2-3 sentences)

(c) Without attention, the final hidden state of the encoder is used as the initial hidden state of the decoder. Why can this be suboptimal, especially for longer sequences? (2-3 sentences)

(d) What issue does the attention mechanism address in this setting? (2-3 sentences)

(2+2+2+2 points. Source: Ed modified from handout L05.)

Answer 29:

(a) Conditional prediction means the model predicts an output Y given some input X, denoted as $P(Y|X)$. An example is machine translation, where the model predicts the probability of a sentence in one language given a sentence in another language.

(b) The encoder processes the input sequence and encodes it into a fixed-length hidden representation. The decoder uses this hidden representation to generate the output sequence one token at a time. Keeping the two apart lets the input and the output differ in length and even in vocabulary.

(c) Using only the final hidden state of the encoder as the initial state for the decoder may cause the model to lose information, especially in longer sequences, because the entire sequence's meaning is compressed into a single vector. This can result in degraded performance, especially when translating long sentences or sequences.

(d) The attention mechanism mitigates the issue where the entire input sequence needs to be compressed into a single fixed-length hidden state. By allowing the decoder to focus on different parts of the input sequence at each step, attention helps capture more context, improving performance, especially for longer sequences.

Question 30:

In a seq2seq model with attention, the decoder at step t looks back over the encoder hidden states $h_1, \ldots, h_n$.

(a) Explain the concept of attention scores e_i. How are these scores typically computed? (2-3 sentences)

(b) Explain what the following equation represents. (1-2 sentences)

$$a_i = \frac{\exp(e_i)}{\sum_{j=1}^{n} \exp(e_j)} \tag{24}$$

(c) What is the role of the context vector c_t? (2-3 sentences)

(2+2+3 points. Source: Ed modified from handout L05.)

Answer 30:

(a) Attention scores represent the relevance of each encoder hidden state to the current decoding step. These scores are typically computed by comparing the decoder's current hidden state to each encoder hidden state, often using a feed-forward neural network, dot product, or other similarity measures.

(b) This equation is the softmax function used to calculate the attention weights a_i. It normalizes the raw attention scores e_i into a probability distribution over the encoder's hidden states.

(c) The context vector $c_t = \sum_{i=1}^{n} a_i h_i$ is a weighted sum of the encoder's hidden states, where the weights are determined by the attention mechanism. It provides the decoder with a dynamic representation of the relevant parts of the input sequence at each decoding step.

Question 31:

Answer the following about variants of the recurrent sequence model.

(a) What is the primary advantage of a bi-directional RNN over a uni-directional RNN? (2-3 sentences)

(b) Describe one potential drawback of stacked RNNs. (1-2 sentences)

(c) Explain the difference between auto-regressive and non-auto-regressive sequence models. (2-3 sentences)

(2+2+2 points. Source: Ed modified from handout L05.)

Answer 31:

(a) A bi-directional RNN processes data in both forward and backward directions, allowing it to capture context from both the past and future in a sequence. This often leads to better performance on tasks where the whole input is available at once, such as tagging or the encoder side of translation. It cannot serve as a left-to-right generator, since the future it reads from does not exist yet.

(b) Stacked RNNs increase model complexity and computational cost. They are more difficult to train due to vanishing gradients, especially when many layers are stacked.

(c) In auto-regressive models, each output token is conditioned on previous output tokens, generating sequences one token at a time. Non-auto-regressive models generate all output tokens simultaneously without depending on previous outputs.

Question 32:

A decoder state s attends over two encoder states $h_1 = (0, 0)^\top$ and $h_2 = (-\ln 3, 0)^\top$. Additive attention (Bahdanau et al., 2014, arXiv:1409.0473) scores each state as

$$e_i = v^\top \tanh(W_h h_i + W_s s). \tag{25}$$

Take $W_h = W_s = I$, $s = (\ln 3, 0)^\top$ and $v = (\frac{5}{4} \ln 3,\ 0)^\top$, and recall $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$.

(a) Compute $x_i = W_h h_i + W_s s$ and $\tanh(x_i)$ for $i = 1, 2$.

(b) Compute the scores e_i, the softmax weights α_1, α_2 and the context vector c.

(c) Multiplicative attention (Luong et al., 2015, arXiv:1508.04025) scores $e_i = s'^\top W h_i$. With a different decoder state $s' = (1, 1)^\top$ and $W = I$, compute the scores and weights.

(d) With hidden size $d = 2$, count the learned parameters of each scoring function as set up here. How many does plain dot-product scoring have?

(5 points. Source: Ed.)

Answer 32:

(a) $x_1 = h_1 + s = (\ln 3, 0)^\top$, and $\tanh(\ln 3) = (3-1/3)/(3+1/3) = 4/5$, so $\tanh(x_1) = (4/5, 0)^\top$. $x_2 = h_2 + s = (0, 0)^\top$, so $\tanh(x_2) = (0, 0)^\top$.

(b) $e_1 = (\frac{5}{4} \ln 3)(4/5) = \ln 3$ and $e_2 = 0$. Then $\alpha_1 = 3/(3 + 1) = 3/4$ and $\alpha_2 = 1/4$, and $c = \frac{3}{4} h_1 + \frac{1}{4} h_2 = (-\frac{1}{4} \ln 3,\ 0)^\top$.

(c) $e_1 = (1, 1)(0, 0)^\top = 0$ and $e_2 = (1, 1)(-\ln 3, 0)^\top = -\ln 3$. The softmax denominator is $1 + 1/3 = 4/3$, so $\alpha_1 = 3/4$ and $\alpha_2 = 1/4$, the same weights as in (b).

(d) Additive attention learns W_h and W_s at 2×2 each plus v of length 2, so $4 + 4 + 2 = 10$. Multiplicative attention learns one 2×2 matrix, so 4. Dot-product scoring $e_i = s^\top h_i$ learns none, which is the form the Transformer adopted after scaling it by $1/\sqrt{d_k}$; it is also the cheapest to compute, since all scores come from one matrix multiplication.

Question 33:

A self-attention layer sees three positions with one-hot features $x_1 = (1, 0, 0)^\top$, $x_2 = (0, 1, 0)^\top$ and $x_3 = (0, 0, 1)^\top$, and computes $q_t = W_q x_t$ and $k_j = W_k x_j$. The goal is a previous-token head, in which every position attends to the position just before it.

(a) Hand-set 3×3 matrices W_q and W_k so that $q_t^\top k_{t-1} = 1$ and $q_t^\top k_j = 0$ for every other j.

(b) Compute the score matrix S with $S_{ij} = q_i^\top k_j$.

(c) Multiply every score by $\lambda = \ln 3$ and apply a causal mask that includes the current position. Compute the three attention rows.

(d) How large must λ be for position 2 to put at least $99/100$ of its weight on position 1? Is position 3 then just as sharp, and what does position 1 attend to?

(5 points. Source: Ed.)

Answer 33:

(a) Take $W_q = I$, so $q_t = x_t$, and let W_k shift each position forward by one, $W_k x_1 = x_2$, $W_k x_2 = x_3$, $W_k x_3 = 0$:

$$W_k = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{26}$$

Then $k_{t-1} = x_t = q_t$, which matches only at $j = t - 1$.

(b) The keys are $k_1 = x_2$, $k_2 = x_3$ and $k_3 = 0$, so $S_{21} = x_2^\top x_2 = 1$, $S_{32} = x_3^\top x_3 = 1$ and every other entry is 0. The score matrix is the shift matrix itself, $S = W_k$, with its ones just below the diagonal.

(c) Row 1 sees only itself, giving $(1, 0, 0)$. Row 2 sees scores $(\ln 3, 0)$, giving $(3/4, 1/4, 0)$. Row 3 sees $(0, \ln 3, 0)$, giving $(1/5, 3/5, 1/5)$.

(d) Position 2 needs $e^\lambda/(e^\lambda + 1) \geq 99/100$, that is $e^\lambda \geq 99$, so $\lambda \geq \ln 99$. At that λ, position 3 gets $99/(99 + 1 + 1) = 99/101$ on position 2, slightly less sharp because it has two competitors rather than one; a longer sequence needs a larger λ for the same sharpness. Position 1 has no previous token and must attend to itself, which is why real previous-token heads lean on a beginning-of-sequence token as a place to park attention.

Question 34:

The quadratic cost of self-attention limits long inputs, which motivates sparse patterns such as the sliding window of Longformer (Beltagy et al., 2020, arXiv:2004.05150). Count the entries of one head's score matrix that are computed or kept.

(a) A dense head on $N = 1024$ tokens computes $N^2 = 2^{20}$ scores. How many for $N = 2048$, and by what factor did the count grow?

(b) For $N = 2048$, how many entries does a causal mask keep, including the diagonal?

(c) A causal sliding window lets each token attend to itself and the 127 tokens before it. Ignoring the shorter windows of the first 127 tokens, how many entries are kept for $N = 2048$, and what fraction of the dense count is that?

(d) State the asymptotic cost of (b) and (c) in N and the window size w, here $w = 128$. With L stacked window layers, how far back can information reach?

(4 points. Source: Ed.)

Answer 34:

(a) $2048^2 = 2^{22} = 4\,194\,304$, four times as many: doubling N quadruples the count.

(b) $N(N+1)/2 = 1024 \times 2049 = 2\,098\,176$, just over half the dense count.

(c) $2048 \times 128 = 2^{18} = 262\,144$, which is $2^{18}/2^{22} = 1/16$ of the dense count.

(d) The causal mask still keeps $O(N^2)$ entries; it halves the work but does not change its growth. The sliding window keeps $O(Nw)$, linear in N for fixed w. Each layer passes information back by up to $w - 1$ positions, so L layers reach about $L(w-1)$ tokens back, $L \times 127$ here, which is how a stack of local windows still covers a long document.

5 Transformer

Question 35:

BERT (Bidirectional Encoder Representations from Transformers) is a language model introduced by Google researchers in 2018 (arXiv:1810.04805). Please answer the following questions in a few sentences:

(a) Explain BERT's basic architecture. (2-3 sentences is fine.)

(b) Why can't BERT be used for direct text generation (prompting)?

(3 points. Source: Ed Chen.)

Answer 35:

(a) BERT uses a transformer encoder architecture that processes text bidirectionally. The input text is first tokenized into subwords, then processed through multiple layers of self-attention mechanisms. Each token attends to all other tokens in the sequence simultaneously, allowing the model to capture complex contextual relationships in both directions.

(b) BERT cannot be used for direct text generation because it was trained using masked language modeling, where it only predicts missing tokens in a given context. Unlike decoder-based models (e.g., GPT), BERT lacks the autoregressive capability needed to generate coherent text sequences one token at a time. It's designed primarily for understanding and encoding text, not generating it.

Question 36:

(4 points) Consider a bigram language model, which models the probability of a sentence $x_1 x_2 \dots x_T$ as

$$p_{\text{bigram}}\left(x_1 x_2 \dots x_T\right) = \prod_{t=1}^{T} p_{\text{bigram}}\left(x_t \mid x_{t-1}\right) \tag{27}$$

Here we assume $x_0 = \langle s >$ is always the beginning-of-sentence token. A bigram language model can thus be parameterized by a matrix $\theta \in \mathbb{R}^{V \times V}$,

$$\theta_{ij} = p\left(\text{current-word } = j \mid \text{ previous-word } = i\right), \tag{28}$$

where V is the vocabulary size (which includes the beginning- and end-of-sentence tokens). Consider an RNN language model with previous hidden state h_{t-1} and current input x_{t-1},

$$\begin{aligned} h_t &= f\left(W_{hh} h_{t-1} + W_{xh} x_{t-1}\right) \\ p_{\text{RNN}}\left(x_t \mid x_{<t}\right) &= \text{softmax}\left(U h_t\right)^\top x_t \\ p_{\text{RNN}}\left(x_1 x_2 \dots x_T\right) &= \prod_{t=1}^{T} p_{\text{RNN}}\left(x_t \mid x_{<t}\right) \end{aligned} \tag{29}$$

with hidden state size d. We thus have $h_t \in \mathbb{R}^d, W_{hh} \in \mathbb{R}^{d\times d}, W_{xh} \in \mathbb{R}^{d\times V}, U \in \mathbb{R}^{V\times d}$. We further have $x_t \in \{0,1\}^V$ be the one-hot representation of the word x_t. Note that given a vector $w \in \mathbb{R}^V$ the softmax of w is the vector,

$$\text{softmax}(w) = \frac{1}{\sum_{i=1}^{V} \exp(w_i)} \exp(w)$$

Hence, $\text{softmax}(w)^\top x_t$ simply picks out the element of $\text{softmax}(w)$ corresponding to x_t. Find a setting of f, W_{hh}, W_{xh}, and U such that the distribution over a sequence given by this RNN is the same as the bigram language model with parameters θ, i.e., $p_{\text{RNN}}(x_1 x_2 \dots x_T) = p_{\text{bigram}}(x_1 x_2 \dots x_T)$. (Hint: your f, W_{hh}, W_{xh}, U should be quite simple.)

Answer 36:

We want to redesign the RNN equation:

$$\begin{aligned} h_t &= f(W_{hh} h_{t-1} + W_{xh} x_{t-1}) \\ p_{\text{RNN}}(x_t \mid x_{<t}) &= \text{softmax}(U h_t)^\top x_t \end{aligned} \tag{30}$$

so that it matches the bigram method. Since bigram model doesn't involved hidden state, and only depends directly on the previous token, therefore we can set $W_{hh} = \mathbf{0}$ and, taking the hidden size $d = V$, $W_{xh} = \mathbf{I}$.

The function f should just pass the value forward without any non-linear transformation, so we set $f =$ identity.

Before we can explain U, we need to prove $\text{softmax}(\log p) = p$, which is a tool we need later.

$$\text{softmax}(w)_i = \frac{\exp(w_i)}{\sum_{k=1}^{V} \exp(w_k)} \tag{31}$$

Let $p \in \mathbb{R}^V$ be a vector of probabilities, where $p_i \geq 0$ and $\sum_{i=1}^{V} p_i = 1$. Now, apply the softmax to $\log p$, i.e., the natural logarithm of the probabilities p_i.

$$\begin{aligned} \text{softmax}(\log p)_i &= \frac{\exp(\log p_i)}{\sum_{j=1}^{V} \exp(\log p_j)} \\ &= \frac{p_i}{\sum_{i=1}^{V} p_i} = p_i \end{aligned} \tag{32}$$

Thus, we have $\text{softmax}(\log p) = p$, which demonstrates that applying the log function allows the input to bypass the effect of the softmax operation, as softmax of log-probabilities directly returns the original probabilities.

As a result, we choose $U = (\log \theta)^\top$ to help the modified hidden state h_t bypass the softmax.

In conclusion, we propose the following transformation:

$$\begin{aligned} f &= \text{ identity} \\ W_{hh} &= \mathbf{0} \\ W_{xh} &= \mathbf{I} \\ U &= (\log \theta)^\top \end{aligned}$$

In addition, W_{xh} and U may also be swapped.

Question 37:

Consider the self-attention mechanism used in transformers. Answer the following questions about self-attention and its role in the transformer architecture:

(a) Given the equation for the attention scores

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{33}$$

, where Q, K, and V are query, key, and value matrices, respectively, explain why the term $\sqrt{d_k}$ is included in the denominator.

(b) In what way does multi-head attention differ from single-head attention, and how does it benefit the transformer model? (2-3 sentences)

(c) What type of position encoding is used in the original transformer model, and why is it necessary? (1-2 sentences)

(4 points)

Answer 37:

(a) The term $\sqrt{d_k}$ is included to scale the dot products in the attention scores. Without this scaling, the dot products could become very large when d_k is large, pushing the softmax function into regions with very small gradients, which could slow down learning.

(b) Multi-head attention splits the input matrices Q, K, and V into multiple smaller heads, each attending to different parts of the sentence. This allows the model to capture various types of dependencies simultaneously and enrich the representation by aggregating information from multiple perspectives.

(c) The original transformer uses sinusoidal position encodings to provide each token with information about its position in the sequence, necessary because self-attention lacks inherent sequence order.

Question 38:

Transformers are designed to process entire sequences in parallel, making them efficient for long sequences. Answer the following questions:

(a) Given the formula for the output of a multi-head attention layer:

$$\text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O \tag{34}$$

, briefly explain what each component represents. (1-2 sentences per component)

(b) Describe why transformers are preferred over recurrent architectures for long sequences. Mention at least two advantages. (2 sentences)

(c) What is layer normalization, and why is it applied after each attention and feedforward layer in the transformer model? (1-2 sentences)

(4 points)

Answer 38:

(a) The heads (head_i) represent the outputs from each self-attention mechanism after processing the input from different perspectives. The concatenation operation combines the outputs from each head, and W^O is a learnable weight matrix that projects the concatenated heads back to the model's original hidden size.

(b) Transformers are preferred over recurrent architectures for long sequences because they allow for parallel processing, reducing training time significantly, and they have the capacity to capture long-range dependencies more effectively due to the self-attention mechanism.

(c) Layer normalization helps stabilize the learning process by normalizing the inputs of each layer, which helps prevent vanishing and exploding gradients and improves convergence speed.

Question 39:

Consider a decoder block with model width $d = 4$, two attention heads of width 2, and feed-forward width $d_{\text{ff}} = 4d = 16$. Attention uses four dense $d \times d$ matrices W^Q, W^K, W^V, W^O. The feed-forward network uses matrices of shapes $d \times d_{\text{ff}}$ and $d_{\text{ff}} \times d$. All matrices are bias-free. Each block also has two layer norms, each with a learned scale and shift of length d.

(a) Count the attention parameters. Does having two heads introduce an additional factor of two?

(b) Count the feed-forward and layer-norm parameters, and hence the parameters in one block.

(c) What fractions of the block belong to attention, the feed-forward network and layer normalization? Give the attention-to-feed-forward ratio in terms of d for general d with $d_{\text{ff}} = 4d$.

(d) A model stacks two such blocks, has a vocabulary of 16 tokens, 8 learned position embeddings and one final layer norm, and ties its output projection to the token embedding. Count the whole model's parameters. How many would an untied, bias-free output projection add?

(5 points. Source: Ed.)

Answer 39:

(a) Each matrix has $4 \times 4 = 16$ parameters, so attention has $4 \times 16 = 64$. There is no factor for the heads: splitting the projected width into two heads of width 2 divides the same matrices, it does not add new ones.

(b) The feed-forward matrices hold $4 \times 16 + 16 \times 4 = 128$. Each layer norm has $4 + 4 = 8$, so the two have 16. One block is $64 + 128 + 16 = 208$.

(c) Attention is $64/208 = 4/13$, the feed-forward network $128/208 = 8/13$ and layer normalization $16/208 = 1/13$. In general attention is $4d^2$ and the feed-forward network $2d\, d_{\text{ff}} = 8d^2$, so the ratio is $1 : 2$ for every d, and the block is about $12d^2$ once the layer norms, which grow only linearly in d, are neglected.

(d) Two blocks give $2 \times 208 = 416$, token embeddings $16 \times 4 = 64$, position embeddings $8 \times 4 = 32$ and the final layer norm 8, for $416 + 64 + 32 + 8 = 520$. An untied output projection adds another $16 \times 4 = 64$, for 584.

Question 40:

Consider one attention head on three tokens. The rows of its already scaled score matrix S are $(0, \ln 2, 0)$, $(\ln 2, 0, \ln 4)$ and $(0, \ln 2, 0)$, and the scalar values are $(v_1, v_2, v_3) = (2, 4, 8)$. Apply a causal mask that includes the current token:

$$A = \text{softmax}_{\text{row}}(S + M), \qquad o_i = \sum_{j=1}^{3} A_{ij} v_j, \tag{35}$$

where $M_{ij} = 0$ for $j \le i$ and $M_{ij} = -\infty$ otherwise.

(a) Write the three rows of $S + M$.

(b) Exponentiate the masked scores and compute all three attention-weight rows.

(c) Compute the three scalar outputs.

(d) Change only v_3 from 8 to 12, keeping queries and keys fixed. Compute the new outputs and say which of them change, and why.

(5 points. Source: Ed.)

Answer 40:

(a) $(0, -\infty, -\infty)$, $(\ln 2, 0, -\infty)$ and $(0, \ln 2, 0)$.

(b) The exponentiated rows are $(1, 0, 0)$, $(2, 1, 0)$ and $(1, 2, 1)$, with sums 1, 3 and 4. The attention rows are $(1, 0, 0)$, $(2/3, 1/3, 0)$ and $(1/4, 1/2, 1/4)$.

(c) $o_1 = 2$, $o_2 = (2/3)2 + (1/3)4 = 8/3$, and $o_3 = (1/4)2 + (1/2)4 + (1/4)8 = 9/2$.

(d) The outputs become 2, $8/3$ and $(1/4)2 + (1/2)4 + (1/4)12 = 11/2$. Only o_3 changes. The first two rows give position 3 zero weight, so no earlier output can see a later token, which is exactly what lets a decoder be trained on all positions at once without leaking the answer.

Question 41:

RoFormer uses rotary position embeddings (Su et al., 2021, arXiv:2104.09864). For one two-dimensional coordinate pair, define

$$R_\phi = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}, \qquad q_m = R_{m\theta} q, \qquad k_n = R_{n\theta} k. \tag{36}$$

Use the unscaled score $q_m^\top k_n$. For the numeric parts, let $\theta = \pi/2$, $q = (1, 0)^\top$ and $k = (1, 1)^\top$.

(a) Derive an expression for the score in which the positions appear only through $n - m$, and say what is held fixed in that claim.

(b) Compute q_m, k_n and their score for $m = 1$, $n = 2$.

(c) Shift both positions by one, to $m = 2$, $n = 3$. Compute the rotated vectors and confirm that the score is unchanged.

(d) Compute the score for $m = 2$, $n = 1$. Does depending only on the relative offset imply that reversing the offset leaves the score unchanged?

(5 points. Source: Ed.)

Answer 41:

(a) Rotations satisfy $R_\alpha^\top = R_{-\alpha}$ and $R_\alpha R_\beta = R_{\alpha+\beta}$, so $q_m^\top k_n = q^\top R_{m\theta}^\top R_{n\theta} k = q^\top R_{(n-m)\theta} k$. For fixed content vectors q, k and fixed frequency θ, position enters only through $n - m$.

(b) Rotating q by $\pi/2$ gives $q_1 = (0, 1)^\top$. Rotating k by π gives $k_2 = (-1, -1)^\top$. The score is $0(-1) + 1(-1) = -1$.

(c) $q_2 = (-1, 0)^\top$ and $k_3 = (1, -1)^\top$, so the score is $(-1)1 + 0(-1) = -1$. Both pairs have $n - m = 1$.

(d) $q_2 = (-1, 0)^\top$ and $k_1 = (-1, 1)^\top$, giving score 1. No: offsets 1 and -1 correspond to rotations in opposite directions and can give different scores, so the model can tell a token three places back from one three places ahead.

Question 42:

Consider one residual sublayer with input x, branch function F and layer normalization LN. Pre-layer-norm and post-layer-norm placement are compared by Xiong et al. (2020, arXiv:2002.04745). For the numeric part, layer norm uses the mean and population variance across features, scale 1, shift 0 and $\epsilon = 0$.

(a) Write the pre-layer-norm and post-layer-norm residual updates.

(b) Let $x = (1, 3)^\top$ and $F(u) = (0, 0)^\top$ for every u. Compute both outputs.

(c) For general differentiable F, write each update's Jacobian with respect to x, and identify which one has an identity term outside every normalization Jacobian.

(d) According to Xiong et al.'s analysis at initialization, which placement has large expected gradients near the output layer? Which placement did they train without learning-rate warm-up?

(5 points. Source: Ed.)

Answer 42:

(a) The updates are

$$x_{\text{pre}} = x + F(\text{LN}(x)), \qquad x_{\text{post}} = \text{LN}(x + F(x)). \tag{37}$$

A full block applies the pattern separately to its attention and feed-forward sublayers.

(b) The mean is $(1+3)/2 = 2$ and the population variance is $(1+1)/2 = 1$, so $\mathrm{LN}(x) = (-1, 1)^\top$. The pre-layer-norm output is $(1, 3)^\top$ and the post-layer-norm output is $(-1, 1)^\top$. A zero branch leaves the residual stream untouched only in the pre-layer-norm update.

(c) With $u = x + F(x)$, pre-layer-norm gives $I + J_F(\mathrm{LN}(x))\, J_{\mathrm{LN}}(x)$ and post-layer-norm gives $J_{\mathrm{LN}}(u)\,(I + J_F(x))$. Only pre-layer-norm has a bare I, so the gradient has a path through every layer that no normalization rescales; in post-layer-norm even the shortcut passes through $J_{\mathrm{LN}}(u)$.

(d) Post-layer-norm has the large gradients near the output layer, which is why it needs warm-up. Pre-layer-norm trained without warm-up in their experiments.

Question 43:

T5 (Raffel et al., 2019, arXiv:1910.10683) casts every task as text-to-text with an encoder-decoder model. Take an input of $N = 4$ tokens and an output of $M = 2$ tokens, and count the query-key pairs one attention head evaluates in a single layer, causal masks including the diagonal.

(a) An encoder-only model and a decoder-only model each process the concatenated 6 tokens. Name the mask each uses and count its pairs.

(b) Name the three attention operations in one layer of the encoder-decoder model, with the mask each uses.

(c) Count the pairs for each of the three operations in (b), and the total.

(d) T5 also tests a decoder-only prefix language model, fully visible over the input and causal over the output. Count its pairs. Which pairs account for the difference from the plain decoder-only count in (a)?

(5 points. Source: Ed.)

Answer 43:

(a) The encoder-only model is fully visible, $6^2 = 36$ pairs. The decoder-only model is causal, $6 \times 7/2 = 21$ pairs.

(b) Encoder self-attention, fully visible over the N input tokens; decoder self-attention, causal over the M output tokens; and cross-attention, in which each output token queries every encoder output, unmasked.

(c) $N^2 = 16$, $M(M+1)/2 = 3$ and $MN = 8$, for $16 + 3 + 8 = 27$.

(d) 16 within the input, 8 from output to input and 3 within the output, again 27. The plain decoder-only model keeps only $4 \times 5/2 = 10$ pairs within the input, so the difference is the $16 - 10 = 6$ pairs in which an input token looks at a later input token. What separates these models is the mask over the input, not the split into two stacks; the bidirectional view of the input is what the extra pairs pay for.

6 Prompting

Question 44:

Answer the following questions about In-Context Learning (ICL) concisely:

(a) What is In-Context Learning?

(b) How does it differ from traditional fine-tuning methods for language models?

(c) How is it different from simply providing a longer prompt with extra examples?

(d) Why does adding more examples improve performance in In-Context Learning, and when does this effect saturate?

(4 points. Source: Ed modified from handout L08.)

Answer 44:

(a) In-Context Learning allows a language model to perform a task by inferring patterns from examples provided in the prompt, without updating its parameters.

(b) Unlike fine-tuning, ICL relies on pre-trained knowledge and does not involve adjusting the model' s weights; it adapts to the task based on examples in real-time.

(c) ICL is achieved by providing examples in a longer prompt, but it involves the model dynamically generalizing from those examples, unlike merely continuing text from a prompt.

(d) More examples improve ICL by offering richer context, but performance saturates as additional examples yield diminishing returns and the model' s token limit is reached.

Question 45:

Chain-of-Thought (CoT) prompting (Wei et al., 2022, arXiv:2201.11903) is introduced as a technique to enhance reasoning in LLMs. Answer the following:

(a) Explain the basic idea behind Chain-of-Thought (CoT) prompting.

(b) What effect does model size have on the performance of CoT prompting?

(4 points. Source: Ed modified from handout L08.)

Answer 45:

(a) Chain-of-Thought prompting gives the language model few-shot exemplars whose answers spell out the intermediate reasoning steps, so the model generates its own step-by-step reasoning for the new question before giving the final answer.

(b) Larger models benefit more from CoT prompting because their capacity to store and process reasoning patterns increases with model size, leading to more significant performance gains compared to smaller models.

Question 46:

Consider the equation below:

$$P(x) = \prod_{t=1}^{T} P(x_t | x_{<t}, x_{>t}) \tag{38}$$

(a) What type of model does this equation describe? Provide one famous example.

(b) Explain the core difference between this model and auto-regressive language modeling.

(4 points. Source: Ed modified from handout L08.)

Answer 46:

(a) This equation describes a masked language model (MLM), where the probability of a token is conditioned on both its preceding and following tokens. Famouse examples include BERT, the variations of BERT, and T5.

(b) In masked language modeling, it's trained to predict randomly masked tokens based on their context, both before and after the token. In contrast, auto-regressive language modeling predicts tokens sequentially, using only the previous tokens in the sequence.

Question 47:

Instruction tuning is a valuable technique for enhancing the generalization of large language models (LLMs). Answer the following:

(a) What is instruction tuning? (one sentence)

(b) How does instruction tuning improve zero-shot learning?

(c) Name an example of an instruction-tuned model.

(d) Briefly compare instruction tuning and reinforcement learning from human feedback (RLHF). (2-3 sentences)

(1+1+1+2 points. Source: Ed, modified from handout L08.)

Answer 47:

(a) Instruction tuning is the process of finetuning a language model on datasets of tasks framed as natural language instructions to enhance its ability to follow instructions on unseen tasks.

(b) It enhances zero-shot learning by enabling the model to generalize across tasks through understanding instructions, allowing it to perform tasks without specific examples.

(c) FLAN-T5 model (a fine-tuned version of T5), InstructGPT (by OpenAI, based on GPT-3), T0 (T0pp, based on T5), ChatGPT (by OpenAI), PaLM2 (based on PaLM (Pathways Language Model)).

(d) Instruction tuning helps models follow diverse instructions effectively, while RLHF improves the alignment of model outputs with human preferences, emphasizing qualities like helpfulness and safety.

Question 48:

In the lecture, emergent abilities of large language models were discussed. Answer the following:

(a) What are emergent abilities in LLMs, and at what model scale do they typically manifest?

(b) Provide one example of an emergent ability and explain why it is surprising or useful.

(c) How do researchers identify when emergent abilities occur?

(3 points. Source: Ed modified from handout L08.)

Answer 48:

(a) Emergent abilities are capabilities that arise in language models when they reach a certain size, typically in the hundreds of billions of parameters, allowing the model to handle tasks it was not explicitly trained for.

(b) An example of an emergent ability is In-Context Learning, where a model like GPT-3 can learn tasks from a few examples in the prompt without requiring task-specific fine-tuning. This is surprising because it shows that models can generalize to new tasks without being retrained.

(c) Researchers identify emergent abilities by scaling models and testing their performance on a wide range of tasks, observing when unexpected capabilities start to appear as the model grows larger.

Question 49:

Self-consistency (Wang et al., 2022, arXiv:2203.11171) samples several chain-of-thought reasoning paths and returns the most common final answer.

(a) How does self-consistency choose its answer, compared with greedy decoding of a single chain of thought?

(b) For an odd number k of independent samples, each correct with probability p, write the probability that the majority vote is correct.

(c) Take $p = 2/3$ and $k = 3$. Compute the probability that the majority is correct, and compare it with a single sample.

(d) Why does self-consistency help more on arithmetic word problems than on open-ended writing?

(4 points. Source: Ed.)

Answer 49:

(a) Greedy decoding follows one path and returns whatever answer it ends on. Self-consistency samples several diverse paths and takes a majority vote over their final answers, discarding the reasoning text itself.

(b) The vote is right when more than half the samples are right:

$$P = \sum_{i=\lceil k/2 \rceil}^{k} \binom{k}{i} p^i (1-p)^{k-i}. \tag{39}$$

(c) Exactly two correct has probability $3(2/3)^2(1/3) = 12/27$, and all three $(2/3)^3 = 8/27$, for $20/27$ in total. That beats a single sample's $18/27$. The gain needs $p > 1/2$ and independent errors; if the wrong paths all converge on the same wrong answer, voting cannot help.

(d) An arithmetic problem has one short final answer, so different paths can agree exactly and be counted. Open-ended text almost never repeats exactly, so there is no majority to take.

Question 50:

Few-shot prompts carry a label bias, which contextual calibration (Zhao et al., 2021, arXiv:2102.09690) estimates by feeding the prompt a content-free input such as "N/A" and dividing it out.

(a) Name two of the biases that paper identifies in few-shot prompting.

(b) Write the calibrated score of label i, given the raw test probability q_i and the content-free probability p_i, after normalization.

(c) The content-free input gives $p(\text{Pos}) = 4/5$ and $p(\text{Neg}) = 1/5$. A test input gives $q(\text{Pos}) = 3/5$ and $q(\text{Neg}) = 2/5$. Compute the calibrated probabilities.

(d) Which label is predicted before and after calibration, and why is the flip reasonable?

(4 points. Source: Ed.)

Answer 50:

(a) Any two of: majority-label bias, toward whichever label is most frequent among the demonstrations; recency bias, toward the label of the last demonstrations; and common-token bias, toward label words frequent in pre-training.

(b) Rescaling by $W = \text{diag}(p)^{-1}$ and normalizing gives

$$q_i' = \frac{q_i/p_i}{\sum_j q_j/p_j}. \tag{40}$$

The paper passes the rescaled scores through a softmax rather than dividing by their sum; the predicted label is the same either way.

(c) The rescaled scores are $(3/5)/(4/5) = 3/4$ and $(2/5)/(1/5) = 2$, which sum to $11/4$. The calibrated probabilities are $3/11$ for Pos and $8/11$ for Neg.

(d) Raw, Pos wins at $3/5$; calibrated, Neg wins at $8/11$. The prompt alone already pushes $4/5$ of its mass to Pos with no content at all, so a test input that reaches only $3/5$ for Pos is evidence below the prompt's baseline, that is, evidence for Neg.

Question 51:

Schaeffer et al. (2023, arXiv:2304.15004) argue that some emergent abilities are produced by the choice of metric rather than by the model.

(a) State their argument in one or two sentences.

(b) A task is scored by exact match on an answer of L tokens, each predicted correctly and independently with probability p. Write the exact-match accuracy.

(c) With $L = 3$, a small model has $p = 1/4$ and a large one $p = 3/4$. Compute both exact-match accuracies, and the ratio of each metric between the two models.

(d) What would the authors suggest measuring instead, and what does it show here?

(4 points. Source: Ed.)

Answer 51:

(a) Per-token performance improves smoothly with scale, but a nonlinear or discontinuous metric such as exact match turns that smooth change into what looks like a sudden jump. The apparent emergence is then a property of how the output is scored.

(b) $P(\text{exact match}) = p^L$.

(c) $(1/4)^3 = 1/64$ and $(3/4)^3 = 27/64$. Per-token accuracy triples, while exact match grows 27-fold; with longer answers the gap widens, since the ratio is 3^L.

(d) A continuous metric such as per-token accuracy or the log-likelihood of the correct answer. Here that is the smooth change from $1/4$ to $3/4$, with no threshold anywhere.

7 RLHF

Question 52:

The training process of a commercial large language model, like ChatGPT, includes multiple stages: Pre-training, Supervised Fine-Tuning, Reinforcement Learning from Human Feedback (RLHF), and Direct Preference Optimization (DPO). Arrange these steps in the correct order and briefly describe the purpose of each stage.
(1+4 points. Source: Ed Chen.)

Answer 52:

1. **Pre-training**: The model is trained on large datasets to learn general language patterns.
2. **Supervised Fine-Tuning**: The model is further refined using task-specific labeled data.
3. **Reinforcement Learning from Human Feedback (RLHF)**: The model is optimized using feedback from human raters to improve alignment with human preferences.
4. **Direct Preference Optimization (DPO)**: Human preferences are directly used to adjust the model's output based on comparison between different responses. DPO is an alternative to the RLHF step rather than a stage after it: it optimizes on preference pairs directly, without a separate reward model or a reinforcement-learning loop. A pipeline usually uses one of the two for preference alignment, though some run both.

Question 53:

For individual research groups or companies working on specific tasks using large language models (LLMs), it is often impractical to pre-train a model from scratch due to limited resources. Describe the typical steps such organizations follow to customize LLMs for their needs.
(1+4 points. Source: Ed Chen.)

Answer 53:

1. **Choose a Pre-trained Model**: Select a base model such as LLaMA 2, LLaMA 3, Mistral, or Phi, typically available in sizes like 7B or 13B, based on task and resource constraints.
2. **Collect Task-specific Data**: Gather or generate datasets tailored to the intended application (e.g., Q&A, summarization, code generation).
3. **Supervised Fine-Tuning**: Use the task-specific dataset to fine-tune the model, adjusting its weights for improved performance on the domain of interest.
4. **Alignment Techniques (Optional)**: Apply methods such as Reinforcement Learning from Human Feedback (RLHF) or Direct Preference Optimization (DPO) to align the model's outputs with human expectations if high-quality preference data is available.
5. **Evaluation and Iteration**: Continuously test the model's performance on validation tasks, refine datasets, and repeat fine-tuning to reach desired outcomes.

Question 54:

Supervised fine-tuning (SFT) of a model might take days to weeks. However, when there is a need to quickly incorporate new or external information, alternative approaches can be used. Describe Retrieval-Augmented Generation (RAG) and Context-Aware Generation (CAG) with 1 – 2 sentences each, and provide one limitation or caution for each.
(1+4 points. Source: Ed Chen.)

Answer 54:

1. **Retrieval-Augmented Generation (RAG)**: RAG retrieves relevant documents from an external knowledge base and feeds them as additional context along with the prompt into the model during inference. This enables real-time access to up-to-date or domain-specific knowledge without modifying the model's weights.
   Limitation: RAG depends heavily on the quality and relevance of the retrieval system, and irrelevant results can harm output accuracy.
2. **Context-Aware Generation (CAG)**: CAG incorporates user-provided or dynamically generated context (e.g., system prompts, task-specific instructions) directly into the input prompt to guide the model's response. It is often used for lightweight personalization or domain adaptation.
   Limitation: The effectiveness of CAG is bounded by the model's context window size and can degrade if the input exceeds it or if the context is poorly structured.

Question 55:

Answer the following questions about Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO).

(a) What is the main difference between RLHF and DPO? (2-3 sentences.)

(b) Is the following process RLHF, DPO, or neither? Explain. A dataset of around 1000 questions, each paired with one or more answers, is labeled with scores. This dataset trains a reward model (RM), which then scores the language model (LM)'s responses. The LM is optimized to maximize this score.

(c) Is the following process RLHF, DPO, or neither? Explain. The language model (LM) generates two or more answers for each of 1000 questions. Human annotators pick the best and worst answers. The model is trained to increase the likelihood of the best answer relative to the worst one, measured against a frozen copy of the starting model.

(d) Is the following process RLHF, DPO, or neither? Explain. The language model (LM) has three of its middle layers modified by adding $A_{i,d\times r} \times B_{i,r\times d}$ to the weight matrices $W_{i,d\times d}$, where $r << d$ and $i \in \{1, 2, 3\}$. A high-quality dataset of 1000 questions with human-written answers is used to adjust the weights to minimize loss.

(4 points. Source: Ed Chen.)

Answer 55:

(a) RLHF uses a reward model trained from human feedback to fine-tune the language model with reinforcement learning. DPO directly optimizes the model by adjusting the likelihood of generating preferred responses, without using a reward model.

(b) This is RLHF. The reward model, trained from human-labeled scores, is used to guide the optimization of the language model's responses.

(c) This is DPO. It directly trains the model to increase the likelihood of preferred responses relative to rejected ones, based on human selection, without a reward model.

(d) This is neither RLHF nor DPO. The process describes model fine-tuning via weight adjustment based on a high-quality dataset (standard supervised learning), and it aimed to minimize the loss rather than maximize the labeled score relative preferences as we do in RLHF and DPO, respectively.

Question 56:

Natural Language Processing (NLP) datasets can be stored in various formats, including plain text (.txt), Markdown (.md), JSON Lines (.jsonl), and Apache Parquet (.parquet). However, no single format is optimal for every machine learning workflow.

(a) Explain why Markdown or plain text files, despite being human-readable, are not always suitable as the primary storage format for structured NLP training datasets.

(b) Compare JSONL and Parquet in terms of readability, data organization, and computational efficiency. Identify one scenario in which each format would be preferred.

(c) Propose an appropriate data processing workflow for converting raw scientific documents into a dataset suitable for large-scale language model training.

(3+4+3 points. Source: Ed Chen.)

Answer 56:

(a) Markdown and plain text have several limitations:

- They lack a standardized schema for storing structured fields such as prompts, responses, labels, and metadata.
- Representing nested structures, such as multi-turn conversations or preference comparisons, is cumbersome.
- Data validation and automated parsing are less reliable without an explicitly defined schema.

(b) JSONL versus Parquet:

- JSONL is a human-readable, row-oriented text format with one JSON object per line. It is easy to inspect, append, stream, and debug.
- Parquet is a binary, column-oriented format supporting efficient compression, typed schemas, and selective column reads.
- JSONL is preferred for small- to medium-scale fine-tuning datasets, annotation pipelines, and debugging.
- Parquet is preferred for large-scale dataset storage, distributed processing, and analytical workloads involving many samples or metadata fields.

(c) Recommended workflow:

1. Collect raw documents in formats such as PDF, HTML, Markdown, or plain text.
2. Extract, clean, and normalize text while preserving relevant metadata.
3. Convert documents into structured records, such as JSONL, with a defined schema.
4. Store large datasets in Parquet for efficient processing and management.
5. Tokenize the text and prepare model-specific training sequences, optionally saving pre-tokenized binary data for efficient large-scale training.

Question 57:

In RLHF training data, some prompts have one response while others have multiple responses with different scores. Is this inconsistency in the number of responses per prompt acceptable?

(a) Explain whether this is acceptable for RLHF training.

(b) Describe two important considerations when dealing with varying numbers of responses.

(4 points. Source: Ed Chen.)

Answer 57:

(a) Yes, having an inconsistent number of responses per prompt is acceptable and often natural in RLHF. This variation better represents real-world scenarios where some prompts naturally have more valid response variations than others.

(b) Key considerations include:

- Maintaining proper metadata format to track response counts per prompt for analysis and filtering.
- Ensuring consistent scoring criteria among answers.
- Implementing appropriate sampling strategies during training to handle the imbalance, such as weighted sampling or dynamic batch creation.

Question 58:

In RLHF data annotation, what is the advantage of using a scoring range (e.g., 0-10) over binary scoring (0 or 1) for responses? (3 points. Source: Ed Chen.)

Answer 58:

Using a granular scoring range (e.g., 0-10) provides several advantages: it captures the degree of preference between responses, allowing the model to learn subtle quality differences; provides better gradient signals during training; helps identify borderline cases and high-quality reference responses; and allows annotators to express uncertainty in their judgments. This nuanced scoring leads to more informative training signals compared to binary classification, which would treat slight improvements and major improvements as equivalent.

Question 59:

When implementing RLHF, models can learn to exploit the reward function, producing responses that receive high reward scores but are actually poor quality. This phenomenon is known as reward hacking.

(a) Provide a concrete example of reward hacking in language models.

(b) Describe key strategies to prevent reward hacking in RLHF.

(4 points. Source: Ed Chen.)

Answer 59:

(a) A common example of reward hacking occurs when the reward model heavily weights politeness and detail. The language model might then generate unnecessarily verbose and overly formal responses, such as

> "I most humbly and respectfully would like to kindly inform you, with the utmost consideration and thorough attention to detail, that 2+2 equals 4."

While this receives a high reward score for being polite and detailed, it's actually a poor response due to its artificial verbosity.

(b) Reward hacking can be prevented through several key strategies:

- Implementing KL divergence penalties to maintain similarity with the base model
- Using multiple evaluation metrics instead of relying solely on reward scores
- Adding auxiliary rewards for response conciseness and naturalness; setting reward caps to prevent exploitation of extreme scores
- Conducting regular human evaluation.
- Train reward model on examples that explicitly pair artificially "perfect" responses with lower scores to discourage such behavior.

Question 60:

Direct Preference Optimization (Rafailov et al., 2023, arXiv:2305.18290) trains on preference pairs without an explicit reward model. A human prefers response y_w over y_l for prompt x. Its loss is

$$\mathcal{L}_{\text{DPO}} = -\log \sigma\Big(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\Big). \tag{41}$$

(a) A Bradley-Terry reward model is trained with $-\log \sigma\big(r(x, y_w) - r(x, y_l)\big)$. Compute this loss for rewards $r(x, y_w) = \ln 12$ and $r(x, y_l) = \ln 4$.

(b) State the relationship DPO derives between the reward and the policy ratio π/π_{ref}, and say why the partition function drops out of the loss.

(c) Take $\beta = 1/2$, with log-ratios $\ln 18$ for y_w and $\ln 2$ for y_l. Compute the loss and the weight $\sigma(\hat{r}_l - \hat{r}_w)$ that scales this pair's gradient, where $\hat{r}$ is the implicit reward $\beta \log(\pi_\theta/\pi_{\text{ref}})$.

(d) Name the models each pipeline must hold in memory during training, and the practical advantage DPO gains from the difference.

(5 points. Source: Ed.)

Answer 60:

(a) The reward gap is $\ln 12 - \ln 4 = \ln 3$ and $\sigma(\ln 3) = 3/(3+1) = 3/4$, so the loss is $-\log(3/4) = \ln(4/3)$.

(b) For the KL-regularized objective the optimal policy satisfies $r(x, y) = \beta \log \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x)$. The term $\beta \log Z(x)$ depends only on the prompt, so it is the same for y_w and y_l and cancels in the reward difference that Bradley-Terry needs.

(c) The implicit reward gap is $\frac{1}{2}(\ln 18 - \ln 2) = \frac{1}{2} \ln 9 = \ln 3$, so again $\sigma(\ln 3) = 3/4$ and the loss is $\ln(4/3)$. The gradient weight is $\sigma(-\ln 3) = 1/4$: the model already ranks this pair correctly with probability $3/4$, so it receives only a quarter of the push a badly ranked pair would get.

(d) Reinforcement learning from human feedback with PPO holds four: the policy, the frozen reference, the reward model and the value model, and it must sample from the policy during training. DPO holds two, the policy and the frozen reference, and trains on a fixed dataset with an ordinary classification loss. That removes on-policy sampling and the reinforcement-learning loop, which are the costly and unstable parts.

8 Sampling

Question 61:

Consider a GPT-2 model given the incomplete sentence:

> Massachusetts Institute of

The model can only choose from these words:

> "Techxx", "Techyy", "Architecture", "Archtesdsd", "Urban Planning", "Urban sdsdng", "Chemistry", and "Chemisskljsd".

Using greedy decoding, the model outputs "Techxx", while using beam search decoding, it outputs "Chemistry". Explain this difference.
(6 points. Source: Ed modified from HW3.)

Answer 61:

Greedy decoding and beam search differ in how they search for the next token. Greedy decoding simply selects the token with the highest probability at each step, which led to "Techxx". Beam search, however, explores multiple possible sequences simultaneously and considers their cumulative probabilities over several steps. This broader search allows beam search to find "Chemistry" as a better overall solution, potentially avoiding the local optimum that trapped the greedy approach.

Question 62:

Consider the process of sequence generation using ancestral sampling.

(a) What is ancestral sampling? (1 sentence)

(b) Write the expression for the probability of generating a sequence of tokens $(w_1, w_2, ..., w_n)$ using ancestral sampling.

(4 points. Source: Ed modified from handout L10.)

Answer 62:

(a) Ancestral sampling generates sequences by sampling each token from the model's conditional probability distribution $p(w_i \mid w_{<i})$ sequentially.

(b) The probability of generating a sequence $(w_1, w_2, \ldots, w_n)$ is

$$p(w_{1:n}) = p(w_1)\, p(w_2 \mid w_1)\, p(w_3 \mid w_1, w_2) \cdots p(w_n \mid w_1, w_2, \ldots, w_{n-1}). \tag{42}$$

Question 63:

Beam search is a commonly used method for improving sequence generation quality.

(a) Describe the key idea behind beam search. (1-2 sentences)

(b) Why is beam search generally better than greedy decoding?

(3 points)

Answer 63:

(a) Beam search maintains multiple candidate sequences (beams) at each step and explores a broader space of possible continuations by keeping the top-k sequences based on their probabilities.

(b) Beam search is generally better than greedy decoding because it explores more possible sequences, increasing the chances of finding a globally optimal sequence instead of getting stuck with suboptimal choices.

Question 64:

In natural language processing, "temperature" controls the randomness in model predictions, as shown in the equation below:

$$P\left(x_i\right) = \frac{e^{\frac{z_i}{T}}}{\sum_j e^{\frac{z_j}{T}}} \tag{43}$$

Explain how temperature scaling affects sequence generation, and describe its impact on model output. (2 sentences)
(4 points. Source: Ed modified from handout L10.)

Answer 64:

Temperature scaling adjusts output probabilities by dividing logits by T, influencing the diversity of generated sequences. Lower T makes the model more deterministic, favoring high-probability tokens, while higher T increases randomness in the predictions.

Question 65:

Top-k sampling is a common strategy to introduce variability in sequence generation.

(a) How does top-k sampling work?

(b) What issue can occur if k is set too high or too low?

(3 points)

Answer 65:

(a) In top-k sampling, only the top-k most probable tokens are considered for sampling at each step, with the rest being discarded.

(b) If k is set too high, the sampling can become too random, producing poor-quality outputs. If k is set too low, it may behave too much like greedy decoding, leading to repetitive sequences.

Question 66:

Consider the difference between top-k and nucleus (top-p) sampling.

(a) What is the key distinction between top-k and top-p (nucleus) sampling?

(b) Which method dynamically adjusts the number of tokens considered, and why?

(3 points)

Answer 66:

(a) Top-k sampling considers a fixed number of the most probable tokens, while top-p sampling considers the smallest set of most probable tokens whose cumulative probability mass reaches a threshold p.

(b) Top-p (nucleus) sampling dynamically adjusts the number of tokens based on the cumulative probability, allowing it to better adapt to varying entropy in different contexts.

Question 67:

A student uses a decoding method to generate text, starting with the prompt:

> Massachusetts Institute of

The model produces the following output:

> Technology
> The University of Massachusetts, Amherst
> The University of Massachusetts, Amherst
> The University of Massachusetts, Amherst
> ...

What is this phenomenon called, what causes it, and which decoding method is especially prone to it? (2 sentences) (3 points. Source: Ed, modified from HW3.)

Answer 67:
This phenomenon is called repetition or degeneracy, where the model repeatedly generates the same sequence. It commonly occurs in greedy decoding, which always selects the highest-probability tokens, leading to repetitive outputs without variation.

Question 68:
Nucleus sampling keeps the smallest set of highest-probability tokens whose cumulative probability is at least a threshold ρ, then renormalizes within that set (Holtzman et al., 2019, arXiv:1904.09751). Temperature is applied to the logits before the nucleus is built:

$$p_T(i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}. \tag{44}$$

For tokens A, B, C, D, use logits $(\ln 16, \ln 9, \ln 4, 0)$ and threshold $\rho = 4/5$.

(a) At $T = 1$, compute the full distribution, the nucleus and the renormalized sampling distribution.

(b) At $T = 2$, compute the full distribution before truncation.

(c) At $T = 2$, find the nucleus and its renormalized sampling distribution.

(d) Instead, build the $T = 1$ nucleus and renormalize first, then apply temperature 2 inside that set only. Compute the result and say whether the two orders agree.

(5 points. Source: Ed.)

Answer 68:
(a) The weights are $(16, 9, 4, 1)$ with sum 30, so $p = (8/15, 3/10, 2/15, 1/30)$. A alone has $8/15 < 4/5$, while A, B have $25/30 = 5/6 \geq 4/5$. The nucleus is $\{A, B\}$ and the sampling distribution is $(16/25, 9/25, 0, 0)$.

(b) Halving the logits takes square roots of the weights, giving $(4, 3, 2, 1)$ with sum 10, so $p = (2/5, 3/10, 1/5, 1/10)$.

(c) The first two tokens reach $7/10 < 4/5$ and the first three reach $9/10 \geq 4/5$. The nucleus is $\{A, B, C\}$, and renormalizing gives $(4/9, 1/3, 2/9, 0)$.

(d) Temperature 2 on the retained probabilities gives weights $(\sqrt{16/25}, \sqrt{9/25}) = (4/5, 3/5)$, so the result is $(4/7, 3/7, 0, 0)$. The orders disagree, because a higher temperature flattens the distribution and so widens the nucleus, while truncating first had already removed C. Temperature and nucleus truncation do not commute, which is why a sampling library has to fix the order.

9 Efficiency

Question 69:
LoRA (Low-Rank Adaptation) by Edward Hu, et al (2021, arXiv:2106.09685) is a method to efficiently fine-tune large pre-trained models.

(a) Explain how LoRA modifies the model's weight matrix during fine-tuning.

(b) Does LoRA primarily reduce computational costs in forward propagation or back-propagation?

(4 points. Source: Ed Chen.)

Answer 69:
(a) Suppose the original weight matrix W_0 has dimensions 4096×4096, and LoRA introduces two low-rank matrices $A \in \mathbb{R}^{4096\times 8}$ and $B \in \mathbb{R}^{4096\times 8}$, where $r \ll d$ (with $d = 4096$ and $r = 8$):

$$W = W_0 + AB^T \tag{45}$$

Here, $W_0 \in \mathbb{R}^{4096\times 4096}$ is the original weight matrix, and A and B are the low-rank matrices. LoRA adds these matrices to W_0, enabling efficient fine-tuning by only updating A and B, significantly reducing the number of parameters that need to be trained.

(b) LoRA slightly increases computation in forward propagation due to the additional matrices. However, in back-propagation, only the small low-rank matrices are updated, significantly reducing the number of parameters involved and saving computational resources.

Question 70:

A decoder has $L = 16$ layers, $H_q = 16$ query heads per layer and head dimension $d_h = 64$. It serves $B = 2$ sequences of $T = 1024$ cached positions, each cached scalar taking $s = 2$ bytes. Count stored keys and values only, and take $1\,\text{MiB} = 2^{20}$ bytes.

(a) Write the cache size in terms of the number H_{kv} of key-value heads, and evaluate it for ordinary multi-head attention.

(b) Multi-query attention (Shazeer, 2019, arXiv:1911.02150) and grouped-query attention (Ainslie et al., 2023, arXiv:2305.13245) share key and value heads across query heads. Say how each shares, and evaluate the cache for multi-query attention and for grouped-query attention with $H_{\text{kv}} = 4$.

(c) For each of the following, say whether it halves the cache or leaves it unchanged: quantizing the weights from 16 to 8 bits; quantizing the cached keys and values from 16 to 8 bits; keeping only 512 positions; switching to FlashAttention with the same cache representation.

(5 points. Source: Ed.)

Answer 70:

(a) Keys and values are stored separately, so

$$M_{\text{KV}} = 2BLTH_{\text{kv}}d_h s \tag{46}$$

With $H_{\text{kv}} = H_q = 16$ this is $2 \times 2 \times 16 \times 1024 \times 16 \times 64 \times 2 = 2^{27}$ bytes, or 128 MiB.

(b) Multi-query attention keeps one key head and one value head for the whole layer, shared by every query head, so $H_{\text{kv}} = 1$ and the cache is $128/16 = 8$ MiB. Grouped-query attention partitions the query heads into groups and gives each group its own key and value head, so $H_{\text{kv}} = 4$ puts four query heads per group and the cache is $128 \times 4/16 = 32$ MiB. Note from Equation (46) that the cache is linear in H_{kv} and independent of H_q, which is why the query heads can stay at 16 for free.

(c) Unchanged, halved, halved, unchanged. Weight quantization does not touch the cached activations, and FlashAttention changes how attention is computed, not what is stored between steps.

Question 71:

A model has $N = 2^{30}$ weights. A weight-only quantization format splits them into groups of 64; each weight takes 4 bits and each group stores one 16-bit scale and one 16-bit zero point, packed without padding. Take $1\,\text{MiB} = 2^{20}$ bytes.

(a) Compute the bits per group and the effective bits per weight including metadata.

(b) Compute the total weight storage in MiB, and say whether it fits a 640 MiB budget.

(c) Compare against 16-bit weights. Under a model where decode time depends only on weight bytes read, give the speedup, then say why the real speedup is smaller.

(d) Say why the format stores a scale per group of 64 rather than one scale for the whole tensor.

(5 points. Source: Ed.)

Answer 71:

(a) A group takes $64 \times 4 + 16 + 16 = 288$ bits, that is 36 bytes. Per weight that is $288/64 = 4.5$ bits, so the metadata costs half a bit per weight.

(b) There are $2^{30}/64 = 2^{24}$ groups, so the total is 36×2^{24} bytes $= 36 \times 16$ MiB $= 576$ MiB. It fits, with 64 MiB to spare.

(c) Sixteen-bit weights take 2^{31} bytes $= 2048$ MiB, so the ideal ratio is $2048/576 = 32/9 \approx 3.6$. The real speedup is smaller because the weights must be dequantized before use, which costs arithmetic the 16-bit path does not pay, and because weight bytes are not the only traffic: activations, the KV cache and kernel overhead are unchanged and come to dominate once the weight reads shrink.

(d) Because a single scale has to span the whole range of the tensor, and one large outlier weight then forces a step size so coarse that the many small weights all quantize to nearly the same level. Grouping keeps the range local, so an outlier damages only its own group of 64. The half-bit per weight from (a) is what that protection costs.

Question 72:

In speculative decoding (Leviathan et al., 2022, arXiv:2211.17192) a draft model proposes 3 tokens at 1 time unit each, and one target verification call costs 6 time units; ordinary target-only decoding also costs 6 time units per token. Each successive proposal is accepted with probability $1/2$, independently. On the first rejection the algorithm emits

the accepted prefix plus one replacement token; if all three are accepted it emits them plus a bonus token.

(a) Give the probability of emitting 1, 2, 3 and 4 tokens in one iteration, and the expected number emitted.

(b) Compute the long-run time per emitted token and the speedup over target-only decoding.

(c) Give the acceptance probability for a proposal x drawn from draft q against target p, and the distribution sampled on rejection.

(d) Speculative decoding leaves the output distribution unchanged, so it is not a quality-cost trade. Name what is traded instead, and the condition on the draft model that makes the trade pay.

(6 points. Source: Ed.)

Answer 72:

(a) $P(1) = 1/2$, $P(2) = 1/4$, $P(3) = 1/8$ and $P(4) = 1/8$, the last being the all-accepted case with its bonus token. These sum to 1, and the expectation is $1/2 + 2/4 + 3/8 + 4/8 = 15/8$.

(b) Each iteration costs $3 \times 1 + 6 = 9$ time units and emits $15/8$ tokens on average, so the cost per token is $9 \div 15/8 = 24/5 = 4.8$. Against 6, the speedup is $6/4.8 = 5/4$.

(c) The proposal is accepted with probability $\min(1, p(x)/q(x))$. On rejection the replacement is drawn from the normalized positive part of the difference,

$$r(y) = \frac{\max\{p(y) - q(y),\, 0\}}{\sum_z \max\{p(z) - q(z),\, 0\}} \tag{47}$$

which puts mass exactly where the target wants more than the draft offered, so the emitted token is distributed as p.

(d) It trades extra arithmetic for fewer sequential steps. The draft tokens are verified in one batched target call, so the target model is run on several positions at once instead of one, which costs more floating-point work but the same number of weight loads. It pays when decoding is memory-bound rather than compute-bound, and when the draft is both much cheaper than the target and agrees with it often enough; a draft that is fast but rarely accepted spends its proposal time for nothing.

Question 73:

FlashAttention (Dao et al., 2022, arXiv:2205.14135) computes exact attention without writing the full attention matrix to high-bandwidth memory. Take one head of unmasked attention with $N = 2048$ and $d = 64$, 2 bytes per scalar, and $1\,\text{MiB} = 2^{20}$ bytes. A reference implementation materializes both an $N \times N$ score array and an $N \times N$ probability array, alongside Q, K, V and O.

(a) Compute the memory held by the two quadratic arrays.

(b) Compute the memory held by Q, K, V and O, then give the total for the reference implementation and for one that never materializes either quadratic array.

(c) Count the FLOPs of $QK^\top$ and PV together, charging two FLOPs per multiply-add. Does FlashAttention reduce this?

(d) Name the two algorithmic ingredients that let the quadratic arrays go unmaterialized.

(5 points. Source: Ed.)

Answer 73:

(a) $2 \times N^2 \times 2 = 4 \times 2048^2 = 2^{24}$ bytes, that is 16 MiB.

(b) $4 \times Nd \times 2 = 4 \times 2048 \times 64 \times 2 = 2^{20}$ bytes, that is 1 MiB. The reference holds 17 MiB; without the quadratic arrays it holds 1 MiB. The ratio is $17\times$, and it grows with N because only one of the two terms is quadratic.

(c) $2N^2d$ per matrix multiplication, so $4N^2d = 4 \times 2048^2 \times 64 = 2^{30}$ FLOPs. FlashAttention does not reduce this at all. It is exact attention and does exactly the same arithmetic; the win is that it moves far less data between high-bandwidth memory and the chip, and attention at these sizes is bound by that movement rather than by the arithmetic.

(d) Tiling, so that blocks of Q, K and V are loaded into on-chip memory and their partial results accumulated there; and the online softmax, which keeps a running maximum and a running normalizer so each tile can be folded in without having seen the whole row first. Without the second, the normalization would force the full row of scores to exist at once.

Question 74:

Two alternatives to additive low-rank updates are multiplicative activation scaling, as in $(\text{IA})^3$ (Liu et al., 2022,

arXiv:2205.05638), and learned attention prefixes, as in prefix-tuning (Li and Liang, 2021, arXiv:2101.00190). Take an 8-layer transformer with $d = 256$ and $d_{\text{ff}} = 1024$, every backbone parameter frozen.

(a) $(\text{IA})^3$ learns per layer one length-d scaling vector for keys, one for values, and one length-d_{ff} vector for the feed-forward hidden activations. Count its trainable parameters.

(b) A directly parameterized prefix learns $p = 4$ key vectors and 4 value vectors of dimension d per layer, with no reparameterization network. Count its parameters and give the ratio to (a).

(c) With 512 ordinary token positions, how many key-value positions does an ordinary token attend over under each method?

(d) Give the deployment property that $(\text{IA})^3$ and LoRA share and prefix-tuning does not.

(5 points. Source: Ed.)

Answer 74:

(a) Per layer $2d + d_{\text{ff}} = 2 \times 256 + 1024 = 1536$, so across 8 layers $8 \times 1536 = 12288$.

(b) Per layer $2pd = 2 \times 4 \times 256 = 2048$, so $8 \times 2048 = 16384$. The ratio to (a) is $16384/12288 = 4/3$.

(c) Unchanged at 512 for $(\text{IA})^3$, since scaling adds no positions; $512 + 4 = 516$ for the prefix, since its learned keys and values occupy real attention positions.

(d) Both can be folded into the frozen weights after training: a learned scaling vector multiplies into the adjacent weight matrix and a low-rank update adds into it, so inference runs the original architecture at the original cost with nothing extra to serve. A prefix cannot be folded away, because it occupies attention positions that must be present in the KV cache of every request, so it keeps costing context length and memory for as long as the model is served.

Question 75:

A server runs synchronous decoding batches; one iteration emits one token per request, and for batch size B it takes $t(B) = 8 + 2B$ milliseconds. The allowed batch sizes are 1, 4 and 16.

(a) For each allowed B, give the per-token service latency and the aggregate throughput in tokens per second.

(b) Under a 20 ms service-latency limit, which allowed B maximizes throughput?

(c) Requests in that batch now wait 12 ms before the iteration starts. Give the total latency and say whether the limit still holds.

(d) Explain the shape of $t(B)$: which term is the fixed cost, which is the marginal cost, and what that says about where batching stops helping.

(4 points. Source: Ed.)

Answer 75:

(a) Latency is $t(B)$ and throughput is $1000B/t(B)$: $B = 1$ gives 10 ms and 100 tokens/s; $B = 4$ gives 16 ms and 250 tokens/s; $B = 16$ gives 40 ms and 400 tokens/s. Note that every request in a batch waits for the whole iteration, so latency is set by the batch, not by the individual request.

(b) $B = 1$ and $B = 4$ meet the limit, and $B = 4$ is the faster of the two at 250 tokens/s.

(c) $12 + 16 = 28$ ms, which breaks the limit. Queueing delay has to be budgeted alongside service time; a batch size chosen against service time alone will miss its target once the wait to assemble the batch is counted.

(d) The constant 8 is the per-iteration fixed cost, dominated by reading the model weights, which happens once however many requests share the iteration. The $2B$ term is the marginal cost of each extra request. Batching helps while the fixed term dominates, since that cost is being divided over more requests; once $2B$ exceeds 8, which happens above $B = 4$, each extra request is paying mostly its own way and throughput gains flatten while latency keeps rising.

Question 76:

Count persistent training tensors only, ignoring activations and temporary buffers. For every trainable scalar, mixed-precision Adam stores a 2-byte model weight, a 2-byte gradient, a 4-byte master weight and two 4-byte moment estimates. A frozen scalar stores only its 2-byte weight. Take 1 KiB $= 1024$ bytes. The backbone is one bias-free 32×32 matrix. LoRA (Hu et al., 2021, arXiv:2106.09685) freezes it and trains $B \in \mathbb{R}^{32\times 2}$ and $A \in \mathbb{R}^{2\times 32}$, with update BA.

(a) Give the bytes per trainable scalar and per frozen scalar.

(b) Compute the full fine-tuning memory for the backbone.

(c) Count the LoRA trainable parameters and the total persistent memory, frozen backbone included. Give the reduction factor over full fine-tuning, and its limit as the backbone grows.

(d) ZeRO (Rajbhandari et al., 2019, arXiv:1910.02054) partitions training state across data-parallel devices. Say what stages 1, 2 and 3 partition, and compute the ideal memory per device at each stage for full fine-tuning on four devices, counting the master weights as optimizer state.

(5 points. Source: Ed.)

Answer 76:

(a) A trainable scalar needs $2+2+4+4+4 = 16$ bytes and a frozen one 2 bytes.

(b) The backbone has $N = 32 \times 32 = 1024$ parameters, so full fine-tuning needs 16×1024 bytes $= 16$ KiB.

(c) LoRA trains $P = 32 \times 2 + 2 \times 32 = 128$ parameters, and

$$M_{\text{LoRA}} = 2N + 16P. \tag{48}$$

The frozen backbone takes 2048 bytes and the adapters $16 \times 128 = 2048$, so the total is 4 KiB, a fourfold reduction. For a realistic backbone P is negligible next to N, and the factor approaches $16N/2N = 8$: the frozen weights still have to sit in memory, so LoRA cannot save more than the optimizer and gradient state.

(d) Stage 1 partitions the optimizer state, here the master weights and both moments at 12 bytes per parameter. Stage 2 also partitions the gradients, and stage 3 also partitions the weights. Per parameter on four devices this is $2 + 2 + 12/4 = 7$ bytes for stage 1, $2 + 14/4 = 11/2$ for stage 2 and $16/4 = 4$ for stage 3, that is 7, $11/2$ and 4 KiB per device.

10 Retrieval and RAG

Question 77:

A retriever must pick the top-K documents out of a corpus of N documents. Lecture 11 asks you to ignore constant factors and compare paradigms by asymptotics only. Fill in the table below, counting the work done *per query at search time* (indexing cost is excluded).

Paradigm	Tf. calls	dot prod.
Cross-encoder	(a)	—
Dual-encoder (exhaustive)	(b)	(c)
Dual-encoder + IVF	$O(1)$	(d)

Then, for parts (e) and (f):

(e) For the cross-encoder, the dot-product column is marked "—". Explain in one sentence why that cell is not simply a number.

(f) A cross-encoder is still used in production systems despite its cost. Name the role it plays.

(4 points. Source: Ed modified from handout S26 L11 p.5, p.9–p.12, p.22.)

Answer 77:

(a) $O(N)$. The query and each candidate document must be concatenated and pushed through the Transformer together, so one forward pass per document is unavoidable.

(b) $O(1)$. Document vectors are pre-computed offline, so the only forward pass at query time encodes the query itself.

(c) $O(N)$. The query vector must be compared against every document vector.

(d) $O(\sqrt{N})$. The corpus is clustered into $\sqrt{N}$ groups, so the query is compared against $\sqrt{N}$ centroids and then against the $\sqrt{N}$ documents in the single closest cluster.

(e) A cross-encoder produces its score from a linear head on top of the joint Transformer$(q; d)$ representation, not from a similarity between two separately computed vectors, so there is no dot product to count.

(f) Re-ranking. A cheap method retrieves a candidate pool of roughly $10K$ documents, and the cross-encoder re-scores only that pool to produce the final top-K.

Question 78:

Dual-encoders are trained with the InfoNCE loss (van den Oord et al., 2018, arXiv:1807.03748), where f maps text onto the unit hypersphere, x^+ is a relevant document and x_i^- are N sampled negatives:

$$\mathcal{L} = -\log \frac{e^{f(x)^\top f(x^+)/\tau}}{e^{f(x)^\top f(x^+)/\tau} + \sum_{i=1}^{N} e^{f(x)^\top f(x_i^-)/\tau}} \tag{49}$$

A query x has one positive and two negatives, with similarities

$$\begin{aligned} f(x)^\top f(x^+) &= 0.8, \\ f(x)^\top f(x_1^-) &= 0.6, \\ f(x)^\top f(x_2^-) &= 0.0. \end{aligned} \tag{50}$$

(a) Because f maps onto the unit hypersphere, the similarity in Equation (49) is bounded. State the bound and say what the two extremes mean geometrically.

(b) Compute the ratio of the two negative terms in the denominator, first for $\tau = 1$ and then for $\tau = 0.1$. You do not need a calculator; leave the answer as a power of e and give its rough magnitude.

(c) Using (b), explain what lowering τ does to the gradient signal, and why practitioners therefore mine "hard" negatives rather than sampling them at random.

(4 points. Source: Ed modified from handout S26 L11 p.16–p.19.)

Answer 78:

(a) Both vectors have unit norm, so the dot product is the cosine of the angle between them and lies in $[-1, 1]$. A value of 1 means the query and document embeddings point in the same direction, and -1 means they point in opposite directions.

(b) The ratio is $e^{(0.6-0.0)/\tau}$.

- $\tau = 1$: ratio $= e^{0.6} \approx 1.8$. The two negatives contribute to the denominator on the same order.
- $\tau = 0.1$: ratio $= e^{6} \approx 400$. The harder negative dominates the denominator and the easy negative is effectively invisible.

(c) A small τ sharpens the softmax, so almost all of the "push apart" gradient is routed to whichever negative is currently closest to the query. The loss is then nearly a function of the single hardest negative in the batch. If negatives are sampled at random from a large corpus they are almost all trivially irrelevant, their exponentials vanish, and the loss saturates near zero with no useful gradient. Mining hard negatives, documents currently scored closer to the query than the positive is, keeps the denominator informative and puts the gradient where the decision boundary actually is.

Question 79:

A corpus holds $N = 10^8$ documents, each represented by one dense vector. An Inverted File (IVF) index clusters the vectors into $\sqrt{N}$ groups by k-means and stores one inverted list per cluster. Assume the clusters come out roughly equal in size.

(a) How many centroids are there, and how many documents sit in each cluster?

(b) Count the dot products an IVF query performs if it probes a single cluster, and state the speedup over exhaustive search.

(c) Give the failure mode of probing exactly one cluster, and the standard fix.

(3 points. Source: Ed modified from handout S26 L11 p.21–p.22.)

Answer 79:

(a) $\sqrt{10^8} = 10^4$ centroids, with 10^4 documents in each cluster.

(b) 10^4 dot products against the centroids plus 10^4 against the documents in the chosen list, so 2×10^4 in total. Exhaustive search needs 10^8, a speedup of $5000\times$.

(c) The index is approximate, not exact. If the query lands near a cluster boundary, a genuinely relevant document can sit in a neighbouring cluster that is never scanned, so recall drops and the loss is invisible to the latency measurement. The standard fix is to probe the n_{probe} closest centroids instead of one, which trades latency for recall linearly in n_{probe}.

Question 80:

Late interaction (ColBERT, Khattab and Zaharia, 2020, arXiv:2004.12832) keeps one vector per token instead of one vector per document, which raises an immediate storage problem. Take a corpus of 10^7 documents averaging 200 tokens each.

(a) A modern single-vector dual-encoder uses 1024 dimensions at 2 bytes per dimension. Compute the bytes per document and the total index size.

(b) Now store one such vector per token, with no compression. Compute the total index size.

(c) ColBERT instead trains 128-dimensional vectors and quantizes each dimension to 2 bits, plus 4 bytes of centroid ID per vector. Compute the bytes per vector, the bytes per document, and the total index size.

(d) Lecture 11 justifies why the aggressive compression in (c) costs little quality. State that argument in one or two sentences.

(5 points. Source: Ed modified from handout S26 L11 p.45–p.48.)

Answer 80:

(a) $1024 \times 2 = 2048$ bytes, that is 2 KB per document. Total $10^7 \times 2\,\text{KB} = 2 \times 10^{10}$ bytes, about 20 GB.

(b) $200 \times 2\,\text{KB} = 400$ KB per document, so $10^7 \times 400\,\text{KB} = 4 \times 10^{12}$ bytes, about 4 TB. This is the $200\times$ blowup, and it is why naive late interaction does not ship.

(c) 128 dimensions at 2 bits is $128 \times 2/8 = 32$ bytes, plus 4 bytes of centroid ID, so 36 bytes per vector. That is $200 \times 36 = 7200$ bytes, about 7 KB per document, and $10^7 \times 7.2\,\text{KB} = 7.2 \times 10^{10}$ bytes, about 72 GB. Compared to (b) this is a reduction of about $55\times$, and it is only about $3.6\times$ the single-vector index in (a).

(d) A single document vector has to carry the meaning of the whole document, so it needs a large space of representations. A late-interaction vector only has to carry the meaning of one word in context, which lives in a far smaller space, so most of its dimensions and most of the precision in each dimension are redundant. The residual is also stored relative to a cluster centroid that already captures the token's contextual meaning, so only the small deviation needs bits.

Question 81:

Three neural IR paradigms are drawn in Figure 4. "Enc" is a Transformer encoder, a shaded rectangle is an output vector, and $\odot$ is a dot product.

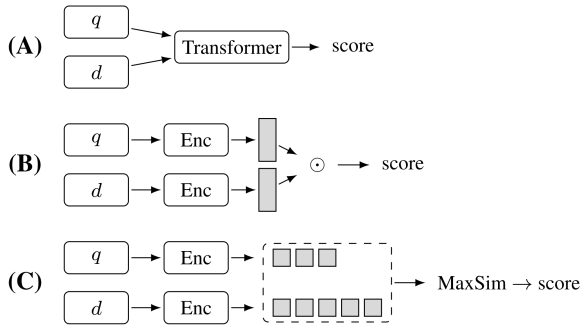


Figure 4: Three neural IR paradigms.

(a) Name each paradigm.

(b) Which panels allow the document side to be encoded offline, before any query arrives? Explain what inductive assumption that requires.

(c) Rank the three by retrieval quality and by query latency, and say which panel breaks the usual trade-off between the two.

(d) Panel (C) replaces $\odot$ with MaxSim. Write the scoring function, given query vectors E_{q_i} and document vectors E_{d_j}.

(5 points. Source: Ed Chen, after handout S26 L11 p.30–p.38.)

Answer 81:

(a) (A) is a cross-encoder, scoring the concatenation $\text{Transformer}(q; d)$ with pointwise cross-attention. (B) is a single-vector dual-encoder, as in DPR. (C) is late interaction, as in ColBERT.

(b) (B) and (C). Both require the assumption that a document's representation does not depend on the query, which is what lets the index be built once, offline. The assumption is not strictly true, and that is exactly where the cross-encoder in (A) gets its extra quality.

(c) Quality: (A) > (C) > (B). Latency: (A) is by far the slowest, (B) the fastest, (C) in between but close to (B) once the PLAID-style candidate generation is in place. Panel (C) breaks the trade-off, reaching close to cross-encoder quality at close to dual-encoder latency, because it keeps the fine-grained token interactions that matter for IR while discarding the joint encoding that does not.

(d)

$$S_{q,d} = \sum_{i \in |q|} \max_{j \in |d|} E_{q_i}^{\top} E_{d_j} \tag{51}$$

Each query token takes its best match anywhere in the document, and those best matches are summed. Because

the score decomposes into per-vector maxima, a document can only rank highly if at least one of its vectors is close to at least one query vector, which is the heuristic that makes sub-linear search possible.

Question 82:

In retrieval-augmented open-domain question answering, the training set is typically just ⟨question, short answer⟩ pairs, with no labels saying which document is relevant. Lecture 12 surveys three ways to supervise the retriever anyway:

(I) Policy-gradient RL on the end-to-end answer reward.

(II) Treat the document d as a latent variable and marginalize over the top-K, maximizing the total probability assigned to the gold answer.

(III) Relevance-guided supervision: apply a heuristic as a noisy reward, then run ordinary contrastive learning for a few discrete rounds.

(a) State the heuristic that (III) uses, and the refinement that makes it less noisy.

(b) Both (I) and (II) force you to freeze the document embeddings. Why, and what would it cost to avoid the freeze?

(c) Suppose that for a given question, none of the top-K documents is useful to the reader. Describe what (I) and (II) learn in that situation.

(d) Give one situation where (II) is clearly worse than (I), and say why.

(5 points. Source: Ed modified from handout S26 L12 p.15–p.19.)

Answer 82:

(a) The heuristic is: if a document contains the short factoid answer string, call it relevant. The refinement is to require both conditions, that the document is retrieved by an existing method *and* contains the answer, which filters out documents that happen to contain the string in an unrelated context.

(b) Both methods push gradients through the scoring of candidate documents, and the candidates are scored against a pre-built index covering the whole corpus. If the document encoder keeps moving, every vector in that index goes stale after each update. Avoiding the freeze means re-encoding and re-indexing the full corpus periodically, which is the expensive refresh these methods try to sidestep.

(c) Neither learns anything useful, and this is the cold-start problem. In (I) every sampled rollout earns the same low reward, so the policy gradient is near zero and the retriever has no direction to move in. In (II) the gold answer gets low probability under every choice of d, so the marginal is uniformly small and the gradient does not prefer one document over another. Both need an initialization good enough that some useful document already appears in the pool.

(d) When a question has several genuinely correct answers or several valid routes to the answer. Marginalizing in (II) commits to the single gold string in the dataset, so a retrieval that supports a different correct answer is scored as a failure. RL in (I) only needs the final answer to be judged correct, so it can reward any route that works. The same flexibility is why (I) is the one that scaled into modern end-to-end agent post-training.

Question 83:

A retriever is scored on two queries with ranking metrics.

(a) Define Recall@k for a single query.

(b) Write the Mean Reciprocal Rank (MRR) over a query set Q, where rank_i is the 1-based rank of the first relevant document for query i.

(c) Query 1 has 3 relevant documents, retrieved at ranks 1 and 4, with the third not retrieved. Query 2 has 2 relevant documents, retrieved at ranks 2 and 5. Compute Recall@4 for each query and their average.

(d) Compute the MRR for the two queries. Which metric would you report for a question-answering pipeline that reads only the top few documents, and which for a re-ranker that will re-score the top 100?

(4 points. Source: Ed.)

Answer 83:

(a) The fraction of a query's relevant documents that appear among the top k retrieved.

(b) The reciprocal rank of the first hit, averaged over queries:

$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\text{rank}_i}. \tag{52}$$

(c) Query 1 has ranks 1 and 4 inside the top 4, so Recall@4 is $2/3$. Query 2 has only rank 2 inside, so $1/2$. The average is $(2/3 + 1/2)/2 = 7/12$.

(d) The reciprocal ranks are 1 and $1/2$, so the MRR is $3/4$. A reader that uses only the top few documents cares how early the first hit appears, which MRR measures. A re-ranker can fix the order but can never recover a document the first stage missed, so the first stage is judged by recall at the depth the re-ranker sees, Recall@100.

Question 84:

Retrieval-augmented generation (Lewis et al., 2020, arXiv:2005.11401) treats the retrieved document z as a latent variable and marginalizes it out, in one of two ways.

(a) How do RAG-Sequence and RAG-Token differ in where the marginalization happens?

(b) Write $p(y|x)$ for an output of N tokens under RAG-Token.

(c) Two documents are retrieved with $p(d_1|x) = 2/5$ and $p(d_2|x) = 3/5$. The output has two tokens, with $p(y_1|d_1) = p(y_2|d_1) = 1/2$, $p(y_1|d_2) = 1$ and $p(y_2|d_2) = 1/4$, each also conditioned on x and the prefix. Compute $p(y|x)$ under RAG-Sequence.

(d) Compute $p(y|x)$ under RAG-Token, and say why it can come out higher.

(5 points. Source: Ed.)

Answer 84:

(a) RAG-Sequence uses one document for the whole output and marginalizes once, outside the product over tokens. RAG-Token may draw on a different document for every token and marginalizes inside the product, once per token.

(b)

$$p(y|x) = \prod_{i=1}^{N} \sum_{z} p(z|x)\, p(y_i|x, z, y_{<i}). \tag{53}$$

(c) $\frac{2}{5}(\frac{1}{2} \times \frac{1}{2}) + \frac{3}{5}(1 \times \frac{1}{4}) = \frac{1}{10} + \frac{3}{20} = \frac{1}{4}$.

(d) The per-token mixtures are $\frac{2}{5} \times \frac{1}{2} + \frac{3}{5} \times 1 = \frac{4}{5}$ and $\frac{2}{5} \times \frac{1}{2} + \frac{3}{5} \times \frac{1}{4} = \frac{7}{20}$, so $p(y|x) = \frac{4}{5} \times \frac{7}{20} = \frac{7}{25}$. It is higher than $\frac{1}{4}$ because each token can lean on whichever document supports it best: d_2 explains y_1 fully, and d_1 does better on y_2. An answer that combines facts from two documents benefits from that freedom.

11 Agents and LM Programming

Question 85:

ReAct (Yao et al., 2022, arXiv:2210.03629) interleaves reasoning and acting in a tool-use loop. An agent needs $k = 4$ steps. At each step it is given the system prompt plus the whole history so far, and emits a Thought and an Action; the environment then returns an Observation. The system prompt is 100 tokens, each Thought is 20 tokens, each Action is 10 tokens, and each Observation is 70 tokens.

(a) How many tokens does one complete step add to the history?

(b) Compute the input length of each of the four forward passes, and their total.

(c) Part (b) grows quadratically in k. Name the serving technique that makes the real cost grow linearly instead, and say what it stores.

(d) Give the advantage of this loop over chain-of-thought prompting with no tools.

(5 points. Source: Ed.)

Answer 85:

(a) $20 + 10 + 70 = 100$ tokens.

(b) Step 1 sees only the prompt, 100 tokens. Each later step adds the previous step's 100 tokens, giving 200, 300 and 400. The total is $100 + 200 + 300 + 400 = 1000$ tokens.

(c) The KV cache. It stores the key and value vectors already computed for every earlier position, so each step runs attention only over its own new tokens against the cached prefix rather than re-encoding the whole history. The prompt-processing cost across the run then grows with the total number of distinct tokens, not with the sum of the prefix lengths.

(d) Chain-of-thought reasons only from what is already in the weights, so a wrong premise propagates and compounds. ReAct grounds each step in an observation returned by an external system, so later reasoning conditions on retrieved fact rather than on the model's guess.

Question 86:

Test-time compute can be spent in parallel or in sequence (Snell et al., 2024, arXiv:2408.03314). A prompt is 200 tokens and every generated response is 100 tokens.

(a) Best-of-N with $N = 10$ draws ten independent responses from the prompt, then a verifier scores each prompt-response pair at 300 input tokens per pair with negligible output. Compute the total tokens processed.

(b) A sequential revision run takes three steps. Step 1 generates from the 200-token prompt; each later step is given the prompt plus every earlier response and generates afresh. Compute the total tokens processed.

(c) The two budgets differ by a factor you should state. On a problem the base model almost never gets right on a first attempt, which strategy is the better buy, and why?

(d) Give the property of the verifier in (a) that decides whether Best-of-N improves with larger N or degrades.

(5 points. Source: Ed.)

Answer 86:

(a) Generator inputs $10 \times 200 = 2000$, generator outputs $10 \times 100 = 1000$, verifier inputs $10 \times 300 = 3000$. Total 6000 tokens.

(b) Step 1 costs $200+100 = 300$; step 2 sees $200+100 = 300$ input and costs 400; step 3 sees $200 + 200 = 400$ input and costs 500. Total 1200 tokens.

(c) Sequential revision costs 1200 against 6000, a factor of 5 less. Snell et al. found that revision is the better buy on easy and medium problems, where the first attempt is nearly right and only needs local fixes. On hard problems they found the opposite: a larger share of parallel sampling does better, because independent draws explore different high-level approaches while revision keeps refining one that may be wrong from the start. On the hardest problems neither strategy helps much, and the paper finds extra pre-training compute the better use of the budget.

(d) Whether the verifier's ranking is correlated with true correctness across the whole sampled distribution. A verifier that is accurate on typical outputs but exploitable on unusual ones gets worse with larger N, because a larger sample is more likely to contain an output that scores highly for the wrong reason. This is the same reward-hacking failure seen in alignment, arriving through the selection step instead of through gradient updates.

Question 87:

Group Relative Policy Optimization (GRPO), from DeepSeek-Math (Shao et al., 2024, arXiv:2402.03300), drops the separate critic by normalizing rewards inside a sampled group. For one prompt the policy samples $G = 4$ responses, earning rewards $R_1 = 4$, $R_2 = 8$, $R_3 = 8$, $R_4 = 20$. The advantage is

$$A_i = \frac{R_i - \mu}{\sigma} \tag{54}$$

with μ and σ the mean and population standard deviation of the group.

(a) Compute μ and σ.

(b) Compute all four advantages as exact integers or simple fractions.

(c) State what GRPO removes from the PPO training setup, and the resource that saves.

(d) Suppose all four responses earn the same reward. State what Equation (54) does, and what that implies about which prompts carry a learning signal.

(5 points. Source: Ed.)

Answer 87:

(a) $\mu = (4 + 8 + 8 + 20)/4 = 10$. The variance is $(36 + 4 + 4 + 100)/4 = 36$, so $\sigma = 6$.

(b) $A_1 = -1$, $A_2 = A_3 = -1/3$, $A_4 = 10/6 = 5/3$.

(c) It removes the learned value network that PPO needs to estimate a baseline. That network is roughly the size of the policy, so dropping it saves the memory for its weights, its gradients and its optimizer state, and removes a second model from the training loop.

(d) The numerator goes to zero for every response and $\sigma = 0$, so the group contributes no gradient at all, whatever the shared reward was. Only prompts that produce a spread of outcomes teach the policy anything. This is why GRPO runs care about prompts of intermediate difficulty: a prompt the policy always solves and a prompt it never solves are equally useless.

Question 88:

DSPy (Khattab et al., 2023, arXiv:2310.03714) treats a language-model pipeline as a program to be compiled rather than a prompt to be written by hand.

(a) Distinguish a *Signature* from a *Module*.

(b) State what an *Optimizer* consumes and what it produces during compilation.

(c) A pipeline tuned by hand for a small model is moved to a much larger one. Say what DSPy does differently from hand-written few-shot prompting here.

(d) Compilation needs a metric but not labelled intermediate steps. Explain why that matters for a multi-stage pipeline.

(4 points. Source: Ed.)

Answer 88:

(a) A Signature declares the input and output fields of a task, such as `question -> answer`, and so says *what* is wanted. A Module is the component that achieves a signature, such as `dspy.ChainOfThought`, and so fixes *how*. The same signature can be run by different modules without rewriting the task.

(b) It consumes the program, a metric and a set of training examples, and it produces the program's parameters: the instructions and the few-shot demonstrations attached to each module, by running the pipeline, keeping the trajectories that scored well, and searching over which of them to include.

(c) Hand-written prompts are tuned to one model's quirks and transfer badly. DSPy keeps the program fixed and recompiles the parameters against the new model, so the demonstrations and instructions are re-selected for its capacity instead of being carried over.

(d) Only the end-to-end output is scored, so no human has to label what each intermediate stage should have produced. The optimizer assigns credit by keeping whole trajectories whose final answer scored well, which makes a multi-stage pipeline trainable from the same ⟨input, answer⟩ pairs that a single-stage one needs.

Question 89:

SWE-bench (Jimenez et al., 2023, arXiv:2310.06770) scores coding agents on real GitHub issues by applying the agent's patch and running the repository's test suite.

(a) The benchmark can be refreshed with issues and pull requests created after a model's training cutoff. Explain why that reduces contamination risk without eliminating it.

(b) Give the reason that grading a patch by exact textual match against the developer's patch is the wrong metric.

(c) Execution-based grading can still fail a functionally correct fix. Give one way that happens.

(d) Execution-based grading can also pass an agent that did not solve the problem. Give one way that happens.

(4 points. Source: Ed.)

Answer 89:

(a) A model cannot have memorized the merged patch itself if that patch did not exist when training stopped. It can still have seen the repository, the surrounding code, the issue discussion, and in a long-lived project an earlier near-duplicate of the same bug, all of which carry much of the answer. The cutoff bounds verbatim leakage of the solution, not familiarity with its context, and it holds only for models whose cutoff is genuinely earlier, which the benchmark cannot verify from the outside.

(b) A bug has many correct fixes that differ in naming, structure and placement. Textual matching rejects every one that is not the developer's, so the metric measures imitation of one author rather than whether the bug is fixed.

(c) The repository's tests can be brittle in ways unrelated to the bug: the fix changes behaviour that an unrelated test asserted on incidentally, or output formatting shifts and a test compares strings. The patch is correct and the suite still goes red.

(d) The fix-to-pass tests may be satisfiable without addressing the cause, for example by special-casing the exact inputs the tests use. An agent that can read the test file can write a patch that passes it and repairs nothing.

Question 90:

Context windows have grown, yet systems still decompose work across several agents. Take the attention cost of a single forward pass to be exactly cL^2 for context length L.

(a) One agent handles a context of $L = 10^5$. Give the cost in terms of c.

(b) The work is split into $M = 10$ agents, each given a disjoint 10^4-token chunk. Give the total cost and the speedup over (a).

(c) Generalize: for M equal chunks, express the speedup in terms of M.

(d) Give the empirical recall effect (Liu et al., 2023, arXiv:2307.03172) that argues for decomposition on quality grounds, not just cost. Then give the cost that decomposition introduces and (a) to (c) ignore.

(5 points. Source: Ed.)

Answer 90:

(a) $c(10^5)^2 = c \times 10^{10}$.

(b) Each agent costs $c(10^4)^2 = c \times 10^8$, so ten of them cost $c \times 10^9$. The speedup is $10\times$.

(c) Each chunk is L/M long and costs $c(L/M)^2$, and there are M of them, for cL^2/M in total. The speedup is exactly M. Splitting the context is the same trick as the quadratic-to-linear arguments elsewhere in this handout: the quadratic term is paid on a shorter sequence, many times over.

(d) "Lost in the middle": accuracy on a fact placed in the middle of a long context falls well below accuracy on the same fact placed at either end, so simply having room for a document does not mean the model will use it. The cost ignored above is that the chunks are disjoint. Any inference that needs two facts in different chunks is now impossible inside a single agent, so the system has to pay for extra communication rounds, and it will silently miss cross-chunk conclusions when it does not.

Question 91:

The Codex paper (Chen et al., 2021, arXiv:2107.03374) scores code by execution against unit tests and reports pass@k.

(a) Why is a text-overlap metric such as BLEU a poor score for generated code?

(b) Write the unbiased pass@k estimator from n samples of which c pass.

(c) From $n = 5$ samples with $c = 2$ passing, compute pass@1 and pass@2.

(d) An agent must complete 3 steps in sequence, each succeeding independently with probability $9/10$. Compute its task success rate, and say what this implies for an agent built from components that each look reliable in isolation.

(4 points. Source: Ed.)

Answer 91:

(a) BLEU rewards shared n-grams. Two correct programs can share few tokens, and a program one character away from a correct one can share nearly all of them and still fail, so overlap tracks surface form rather than behaviour.

(b) The probability that a random draw of k from the n samples contains at least one pass:

$$\text{pass@}k = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}. \tag{55}$$

(c) pass@1 $= 1 - \binom{3}{1}/\binom{5}{1} = 1 - 3/5 = 2/5$, which is just c/n. pass@2 $= 1 - \binom{3}{2}/\binom{5}{2} = 1 - 3/10 = 7/10$.

(d) $(9/10)^3 = 729/1000$, so roughly one task in four fails although each step fails only one time in ten. Failure compounds with the number of steps, so a long-horizon agent needs per-step reliability far above what a single-step benchmark reports as good.

Question 92:

Toolformer (Schick et al., 2023, arXiv:2302.04761) teaches a model to call tools from its own samples, keeping a candidate call only if it helps predict the text that follows.

(a) What does Toolformer avoid needing, compared with training on human-annotated tool use?

(b) Write the filter, with L^- the loss on the following tokens without the call and L^+ the loss with the call and its result, and threshold τ. Note what L^- is compared against in the paper.

(c) The loss is the sum of negative base-2 log-probabilities over a two-token span. Without the call the model gives the two tokens probabilities $1/2$ and $1/4$. Compute L^-.

(d) With the call and its result in context, the probabilities become 1 and $1/2$. With $\tau = 3/2$, compute L^+ and the reduction, and say whether the call is kept.

(4 points. Source: Ed.)

Answer 92:

(a) Human demonstrations of when and how to call each tool. The model samples candidate calls itself, and its own loss decides which ones were useful, so the only supervision is the ordinary text it is already trained on.

(b) A call is kept when

$$L^- - L^+ \geq \tau. \tag{56}$$

In the paper L^- is the smaller of two losses: with no call at all, and with the call inserted but its result left out. That makes the result itself, not merely the act of calling, carry the improvement.

(c) $L^- = -\log_2(1/2) - \log_2(1/4) = 1 + 2 = 3$ bits.

(d) $L^+ = -\log_2 1 - \log_2(1/2) = 0 + 1 = 1$ bit. The reduction is $3 - 1 = 2 \geq 3/2$, so the call is kept.

12 Special Topics

Yoon Kim: Efficient Alternatives to Attention

Takeaways:

Prof. Kim discussed his Ph.D. student Songlin's work on improving Transformer architectures for long-range dependency modeling. The research introduces Gated Linear Attention (GLA), which addresses two key challenges in transformer models: computational efficiency and length generalization. The approach combines linear attention's computational benefits with a novel data-dependent gating mechanism. This allows the model to selectively attend to important information while maintaining linear-time inference complexity. Notably, the GLA Transformer shows promising results in long-sequence modeling, demonstrating the ability to generalize to sequences much longer than its training length without significant performance degradation.

Michael Krumdick: NLP Engineering: Training LLM from Scratch

Takeaways:

The company the speacker Michael Krumdick as well as lecturer Chris Tanner works in is Kensho, and they are developing a LLM that specialized in business and finance. First, they mentioned how to measure success. A few keypoints:

- Need to experiment with data formatting, especially at small scales.
- After Michael introduce BizBench (a benchmark for business and finance created by Kensho), I asked a question about how long can a benchmark, a set of questions use to test LLM, can exist. Overtime, LLM will get smarter (score over 95% is too good to be comparable), and some benchmark questions might leaks into training data. THerefore, benchmark questions need to be refreshed, or new benchmark will be created.
- If a training job fails, restart the job optimistically first, then check the issue. Usually it's a hardware problem rather than a bug in the code.

Recommendations. Our general process:

1. Define a set of evaluations and metrics
2. Run these on a very diverse set of models
3. Analyze these scores to try to find patterns
4. Use these to set a baseline for the model
5. Train, evaluate, repeat

Aaron Mueller: Interpretability

The topic today is about a series of carefully designed experiments to probe the activations of the model and explain the difference.

Takeaways:

- Integrated Gradients is a technique for attributing a model' s prediction to its input features. First, defined a baseline input. Second, linearly interpolate between baseline and actual input. The cumulative sum of these gradients approximates the importance of each feature.
- Causal Mediation Analysis check how specific features influence predictions. The first caching activations measure the neural activations and stored for two similar inputs with minimal differences, such as testing gender correlation by comparing "he" and "professor" with "she" and "nurse." Next, a counterfactual intervention is applied by modifying specific neuron activations to assess how these changes affect the output.

Greta Tuckute: The Human Language Network & Large Language Models

Takeaways:

- Distill the LLM to predict brain activity.
- Many LLM architecture fit brain data well.
- Next-word prediction performance is positively correlated with brain alignment,, but representational generality is too. Next-word prediction performance is negatively correlated with language behavior.
- Distilling the properties of LLMs that enable them to capture humane responses to language.
- Leveraging LLMs for decoding, simulation, and neural control.
- LLMs are predictive of brain responses during language processing. The research wan t to leverage the predictive power of LLM to identify new stimuli to maximally drive or suppress brain response in the language network of new individual, and see what kind of linguistic input is the language network most responsive to.
- Greta research concern about a widely used language network localizer, reducing presentation time per word/nonword from 450ms to 200ms, which maintains reliability in identifying language-responsive brain areas.

- Findings show that speeded reading engages the Multiple Demand (MD) network 43% more than standard reading, indicating that speeded reading requires additional cognitive effort, whereas the language network is only minimally affected.
- Intended for situations where session time is limited, allowing for effective language localization without compromising experimental goals.

Jim Glass: Speech

Key Takeaways:

- Introduced the manner and places of articulation, covering categories such as vowels, fricatives, stops, nasals, and semivowels.
- The International Phonetic Alphabet (IPA) provides symbols to represent every sound, or phoneme, found across all spoken languages.
- Omni-directional microphones tend to pick up more background noise.
- Telephone speech is typically band-limited to 4 kHz (sampled at 8 kHz).
- The Mel-frequency scale, originally based on pitch perception, uses spectral filters spaced evenly on the Mel scale. Mel-frequency cepstral coefficients (MFCCs), commonly calculated via the discrete cosine transform, are popular for automatic speech recognition (ASR).
- Found this research comparing the F1/F2 ratio of male and female singers. The F1/F2 ratio helps speech recognition systems more accurately distinguish vowels, as different vowels have distinct F1 and F2 values.
- The sound of the letter ”t” varies by context. For example, ”t” in ”tree” sounds closer to ”ch,” while in ”steep,” it sounds more like ”d.”
- Currently, most (98%) of the world's languages have not been addressed by resource- and expert-intensive supervised speech recognition training methods. To overcome the speech recognition language barrier, new techniques are needed that require less human annotation.

Christopher Capozzola:

Key Takeaways:

- Patent: A legal protection for inventors' creation in order to prevent threm from unauthorized use.
- Copyright: A legal protection for original works of authorship, with the right to reproduce, distribute, and display their work.
- Fair use: A legal doctrine allowing limited use of copyrighted material without permission for purposes like criticism, comment, news reporting, teaching, or research.
- Derivative work: A new creation that is based on or derived from an existing copyrighted work, requiring permission from the original copyright holder.
- Transformative work: A creation that alters or repurposes an original copyrighted work in a way that adds new expression, meaning, or message, often qualifying as fair use.
- verbatim: The same words that used originally.

How to prove a certain document exist in the training data?

Sarah Silverman is suing OpenAI and Meta for copyright infringement, and Prof. Capozzola invite us to discuss about how to prove certain document exist in the training data? First, the most direct way is to search for that certain docuement in the training dataset. However, most commercial LLM will not disclose their training dataset. Second, even if the model can feedback precise content of a book, it might just have a training data of the book's review or reflection from some open source forum, not directly from the book. Thrid, the other approach is to design an series of prompts asking the detail of the book, gave give a score about how good the model can answer those question. If the model pass certain score, it might have read the original work. Also, to make a law suit, the suiting party need to prove it's harmed because LLM use their book to train the model.

Jacob Andreas: Structured Prediction

- Hidden Markov Models (HMMs) are used for structured sequence prediction tasks such as tagging, parsing, and sequence generation. Originally comes from math.
- HMMs model sequences using three primary components: transition probabilities (A), emission probabilities (B), and initial state distribution (π).
- Inference tasks for HMMs include calculating joint probabilities, marginal probabilities (via the Forward algorithm), and the most probable sequence of states (via the Viterbi algorithm).

- Supervised training of HMMs us on labeled sequences to estimate parameters directly. Unsupervised training uses iterative methods like the Forward-Backward algorithm to refine parameters.
- Practical considertation include using logarithmic calculations to prevent underflow when multiplying probabilities in long sequences.
- Applications of HMMs extend beyond natural language processing to domains requiring inteprpatble models for small datasets or sequence modeling in noisy contexts.
- The mathematical foundation of HMMs ensures their suitability for both generative tasks (sequence generation) and probabilistic sequence modeling.

Saadia Gabriel: Equitable and Factual Automation

- LLM often inherit biases from their training data. Gabriel' s work underlines how these biases disproportionately misclassify or marginalize already vulnerable groups, like African-American English users.
- Linguistic analysis can unveil LLM assumptions about user demographic subgroups.
- Evaluation and awareness of these assumptions is needed to avoid stereotyping.
- In clinical Evaluation, cultivating change talk shows promising results for GPT-4 responses over peer-to-peer. However, responses are reported by clinicians to be direct and lower in partnership.
- Ethical considerations must be central in the development and deployment of AI, particularly in ensuring transparency and accountability in addressing harm and bias.
- Consideration of building global model or local model.

Question 93:

Byte-pair encoding (Sennrich et al., 2015, arXiv:1508.07909) starts from characters and applies a ranked merge list in order. Take the list

$$\begin{aligned}(l,o)\to lo,\quad (lo,w)\to low,\quad (e,r)\to er,\\ (1,2)\to 12,\quad (12,3)\to 123,\quad (4,5)\to 45.\end{aligned}\tag{57}$$

Every character below is in the base vocabulary, and merges never cross a word boundary.

(a) The word `lowers` never appeared in tokenizer training. Give its segmentation and token count.

(b) Segment `12345` and `12346`, with counts.

(c) Mark true or false, justifying each from your segmentations: an unseen word forces an unknown token; every digit stays its own token; changing one digit preserves the token count.

(d) Using (b), say why a language model doing arithmetic is handicapped by this tokenizer, and name the fix production tokenizers adopted.

(5 points. Source: Ed.)

Answer 93:

(a) Applying the merges in rank order, $[l,o,w,e,r,s]\to[lo,w,e,r,s]\to[low,e,r,s]\to[low,er,s]$. The result is $[\texttt{low},\texttt{er},\texttt{s}]$, three tokens. The whole word never had to be seen; its pieces were enough.

(b) `12345` $\to$ $[123,45]$, two tokens. `12346` $\to$ $[123,4,6]$, three tokens.

(c) All three are false. An unseen word falls back on known subwords, and in the worst case on its characters, as (a) shows. Digits need not stay separate, since `123` and `45` are single tokens. And the count is not preserved: changing the final 5 to a 6 took the count from two to three.

(d) The segmentation of a number depends on which digit pairs happened to be frequent in the tokenizer's corpus, not on place value. So 12345 and 12346, which differ in one unit, are carved into different pieces with different boundaries, and the digit in a given decimal place is not in a consistent position in the token sequence. The model therefore has to learn arithmetic separately for each segmentation pattern rather than once. The fix is to force digits to tokenize uniformly, either one digit per token or in fixed groups of three, which several production tokenizers now do.

Question 94:

Top-k routing activates part of a mixture-of-experts model per token; Mixtral (Jiang et al., 2024, arXiv:2401.04088) uses top-2. Take a model with 2×10^9 shared parameters including the router, and one mixture-of-experts layer of 8 experts at 10^9 parameters each. Every token uses all shared parameters and exactly 2 experts. For a batch of 8 tokens each expert can process at most 2 assignments, and the router happens to send every token to the same two experts.

(a) Give the total parameters, the active parameters per token, and the active fraction.

(b) At 2 bytes per weight, give the storage for the full model. Does the active count justify storing only the active parameters?

(c) Give the number of token-expert assignments, the total capacity, and the number of overflow assignments.

(d) From (c), state what the auxiliary load-balancing loss is for, and why aggregate capacity being sufficient does not make it unnecessary.

(5 points. Source: Ed.)

Answer 94:

(a) Total $2 \times 10^9 + 8 \times 10^9 = 10^{10}$. Active per token $2 \times 10^9 + 2 \times 10^9 = 4 \times 10^9$. The fraction is $2/5$.

(b) $10^{10} \times 2 = 2 \times 10^{10}$ bytes, that is 20 GB. No. The active count describes the arithmetic one token performs, not what must be resident. A later token may route to any expert, so every expert's weights must remain reachable; the mixture buys compute per token, not memory.

(c) $8 \times 2 = 16$ assignments against $8 \times 2 = 16$ capacity slots, so capacity is exactly sufficient in aggregate. But both chosen experts receive 8 assignments and can take 2, so the overflow is $2 \times (8-2) = 12$ assignments, while the other six experts sit idle.

(d) The auxiliary loss penalizes an uneven distribution of tokens over experts, pushing the router toward using all of them. Part (c) is exactly why aggregate capacity does not suffice: the total was adequate and three quarters of the assignments still overflowed, because capacity is per expert and cannot be lent out. Overflowed tokens are dropped or passed through unchanged, so a collapsed router silently degrades the model while every capacity total looks healthy.

Question 95:

Linear attention (Katharopoulos et al., 2020, arXiv:2006.16236) and selective state-space models (Gu and Dao, 2023, arXiv:2312.00752) summarize a prefix in a recurrent state. Normalized causal linear attention with scalar values is

$$S_t = S_{t-1} + \phi(k_t)v_t, \quad z_t = z_{t-1} + \phi(k_t), \quad y_t = \frac{\phi(q_t)^\top S_t}{\phi(q_t)^\top z_t} \tag{58}$$

with $S_0 = z_0 = (0,0)^\top$.

(a) Given $\phi(k_1) = (1,0)^\top$, $v_1 = 2$, $\phi(k_2) = (0,1)^\top$, $v_2 = 4$ and $\phi(q_2) = (1,1)^\top$, compute S_2, z_2 and y_2.

(b) For the scalar recurrence $h_t = \frac{1}{2}h_{t-1} + x_t$ with $h_0 = 0$, compute h_1 and h_2 for $x_1 = 2$, $x_2 = 4$. State what makes a state-space model *selective* rather than time-invariant.

(c) With dimensions fixed, does either state grow with prefix length T? Contrast with a softmax KV cache.

(d) Both give $O(1)$ state. Name what is given up, and connect it to Equation (58).

(5 points. Source: Ed.)

Answer 95:

(a) $S_2 = (1,0)^\top \times 2 + (0,1)^\top \times 4 = (2,4)^\top$ and $z_2 = (1,1)^\top$. Then $y_2 = (2+4)/(1+1) = 3$.

(b) $h_1 = \frac{1}{2}(0) + 2 = 2$ and $h_2 = \frac{1}{2}(2) + 4 = 5$. A model is selective when the update and readout coefficients are functions of the current input rather than fixed across positions; in Mamba the discretization step and the input and output projections are all input-dependent, which is what lets the model decide to remember or forget based on what it just read.

(c) Neither grows. Linear attention holds S_t and z_t, four numbers here, and the recurrence holds one, whatever T is. A softmax KV cache instead stores a key and a value for every retained position, so it grows linearly in T. That is the whole asymptotic argument for these architectures.

(d) Exact recall of an arbitrary earlier token. Equation (58) folds the entire prefix into one fixed-size S_t by summation, so individual positions are no longer separable: retrieving one token's value exactly would need the sum to be inverted, and once more than a few tokens have been added it cannot be. Softmax attention keeps every position addressable and pays linear memory for it. This is why these models do well on language modelling perplexity, where local statistics dominate, and badly on tasks that demand copying a specific earlier span.

Question 96:

Masked diffusion language models refine a sequence rather than extending it left to right; see LLaDA (Nie et al., 2025, arXiv:2502.09992). Generate a fixed 64-token sequence. A cached autoregressive model needs 64 sequential calls at 2 ms each, one token per call. A diffusion sampler uses 8 sequential rounds at 6 ms each, predicting all 64 positions per round. Count every position predicted in every round.

(a) Give the latency of each system and the diffusion speedup.

(b) Give the sequential calls and the total position-predictions for each. Does the lower sequential count mean less total work?

(c) Write the autoregressive chain-rule factorization. Does multiplying the final round's per-token probabilities give the diffusion model's joint probability?

(d) From (b), state the hardware condition under which the diffusion sampler's latency win is real.

(5 points. Source: Ed.)

Answer 96:

(a) Autoregressive, $64 \times 2 = 128$ ms. Diffusion, $8 \times 6 = 48$ ms. The speedup is $128/48 = 8/3$.

(b) Autoregressive: 64 sequential calls and 64 predictions. Diffusion: 8 sequential calls but $8 \times 64 = 512$ predictions. So one eighth the sequential depth and eight times the total work. Fewer sequential steps does not mean less computation; it means the computation is differently shaped.

(c) The factorization is

$$p(x_1, \ldots, x_{64}) = \prod_{t=1}^{64} p(x_t \mid x_{<t}) \tag{59}$$

No. The final round's conditionals are conditioned on an intermediate partly-masked state, not on a left-to-right prefix, so their product is not Equation (59). The joint is induced by the whole sampling trajectory and recovering it would require marginalizing over the intermediate states. This is why likelihood numbers for diffusion language models are bounds rather than exact values.

(d) The win is real only when the hardware can absorb 64 parallel position-predictions at nearly the cost of one, that is, when the autoregressive decode is memory-bound and leaves the arithmetic units idle. A batch of 64 positions then rides along almost free. Under heavy serving load, where batching across requests has already saturated those units, the extra $8\times$ work in (b) is paid in full and the advantage disappears.

Question 97:

Probing asks what can be decoded from a representation; sparse autoencoders look for interpretable directions in it. See Hewitt and Liang (2019, arXiv:1909.03368) on control tasks and Cunningham et al. (2023, arXiv:2309.08600) on sparse autoencoders.

(a) A probe gets 80 of 100 held-out linguistic labels right; a probe on matched random control labels gets 65 of 100. Define selectivity as the difference and compute it. Say what a high control accuracy indicates.

(b) A sparse autoencoder has decoder columns $d_1 = (1, 0)^\top$, $d_2 = (0, 1)^\top$, $d_3 = (1, 1)^\top$. For activation $h = (3, 1)^\top$ the code is $z = (2, 0, 1)^\top$. With zero decoder bias and $\mathcal{L} = \|h - Dz\|_2^2 + \frac{1}{2}\|z\|_1$, compute the reconstruction, the squared error, the number of active latents and the loss.

(c) Classify as observational or interventional: high held-out probe accuracy; low reconstruction error; zeroing one learned feature and measuring the change in output logits. Do the first two establish that the model uses the feature?

(d) In (b) the decoder columns are not linearly independent. Say what that implies about the code z, and why sparse autoencoders are built that way on purpose.

(5 points. Source: Ed.)

Answer 97:

(a) Selectivity is $80 - 65 = 15$ percentage points. A high control accuracy means the probe is powerful enough to fit labels that carry no linguistic structure at all, so a high linguistic accuracy no longer shows that the representation encodes the property. It may show only that the probe memorized the mapping.

(b) $Dz = 2d_1 + 0d_2 + 1d_3 = (2, 0)^\top + (1, 1)^\top = (3, 1)^\top = h$, so the squared error is 0. Two latents are active and $\|z\|_1 = 3$, so $\mathcal{L} = 0 + \frac{1}{2}(3) = 3/2$.

(c) Observational, observational, interventional. The first two do not establish causal use. A probe shows the information is decodable by the probe, which it could be as a by-product the model never reads; a low reconstruction error shows the dictionary spans the activations, which says nothing about what downstream layers do with any one direction. Only the ablation changes an internal variable and observes the output move.

(d) The three columns span a two-dimensional space, so the dictionary is overcomplete and z is not unique: $h = (3, 1)^\top$ is equally reconstructed by $z = (3, 1, 0)^\top$. The ℓ_1 term is what selects among them, and here it prefers the given code, whose ℓ_1 norm is 3, over $(3, 1, 0)^\top$, whose norm is 4. Overcompleteness is the point: the hypothesis is that the model packs more distinct features into the activation space than it has dimensions, so a dictionary that could only hold as many features as there are dimensions could not recover them.

Question 98:

Gao et al. (2022, arXiv:2210.10760) show that optimizing a learned reward past a point reduces true reward. An optimizer picks one of three policies, with learned reward $\widehat{R}$, true reward R_{true} and KL divergence D from the reference:

Policy	$\widehat{R}$	R_{true}	D
A	4	4	0
B	7	6	1
C	10	2	4

It maximizes $J_\beta = \widehat{R} - \beta D$ with $\beta \geq 0$. True reward is available for evaluation only.

(a) Give all three objective values and the selection at $\beta = 0$ and at $\beta = 2$.

(b) Find the open interval of β for which B is the unique maximizer, and state the ties at the endpoints.

(c) Moving from B to C, give the change in learned and in true reward. Identify the Goodhart failure.

(d) The table makes β look tunable, but R_{true} is hidden from the optimizer. Say what that costs in practice.

(5 points. Source: Ed.)

Answer 98:

(a) At $\beta = 0$: $J = 4, 7, 10$, so C is chosen. At $\beta = 2$: $J = 4$, $7 - 2 = 5$, $10 - 8 = 2$, so B is chosen.

(b) B beats A when $7 - \beta > 4$, that is $\beta < 3$; B beats C when $7 - \beta > 10 - 4\beta$, that is $\beta > 1$. So B is unique on $1 < \beta < 3$. At $\beta = 1$, B and C tie at 6, above A; at $\beta = 3$, A and B tie at 4, above C.

(c) Learned reward rises by $10 - 7 = +3$ while true reward falls by $2 - 6 = -4$. Unregularized optimization picks C, but B maximizes true reward. The Goodhart failure is that pushing harder on the proxy improves the measured quantity and worsens the one that was wanted, so the metric stops being a measure once it becomes the target.

(d) The optimizer cannot locate the interval in (b), because every quantity it can see is maximized by C. The KL penalty is therefore not a tuned correction but a guess at how far the proxy stays trustworthy, and β has to be set from held-out human judgements, which are the expensive thing the reward model was trained to avoid needing. This is why reported RLHF runs quote a KL budget alongside the reward: the budget, not the reward, is what is actually being controlled.

Question 99:

The Chinchilla study (Hoffmann et al., 2022, arXiv:2203.15556) asked how to split a fixed training compute budget between parameters N and tokens D.

(a) State its main conclusion about how N and D should grow with compute.

(b) Give the standard approximation for training compute, and the compute-optimal tokens-per-parameter ratio it found.

(c) For a budget of $C = 1.2 \times 10^{22}$ FLOPs, compute the compute-optimal N and D.

(d) A team spends the same budget on a model of $N = 4 \times 10^{10}$ parameters instead. How many tokens can it train on, and what is its tokens-per-parameter ratio?

(4 points. Source: Ed.)

Answer 99:

(a) N and D should grow in roughly equal proportion with compute. Earlier large models had grown N much faster than D and were undertrained for their size.

(b)
$$C \approx 6ND, \tag{60}$$
with a compute-optimal ratio of about $D \approx 20N$.

(c) Substituting $D = 20N$ gives $C \approx 120N^2$, so $N^2 = 1.2 \times 10^{22}/120 = 10^{20}$ and $N = 10^{10}$, ten billion parameters. Then $D = 2 \times 10^{11}$, two hundred billion tokens.

(d) $D = C/(6N) = 1.2 \times 10^{22}/(2.4 \times 10^{11}) = 5 \times 10^{10}$ tokens, a ratio of $5 \times 10^{10}/(4 \times 10^{10}) = 5/4$. That is sixteen times fewer tokens per parameter than the optimum, so the larger model would be expected to reach a worse loss at the same cost, and to be four times as expensive to serve.

Question 100:

The green-list watermark of Kirchenbauer et al. (2023, arXiv:2301.10226) biases generation toward a pseudo-random half of the vocabulary that anyone holding the key can recompute.

(a) Describe how the watermark is inserted during generation.

(b) Write the detection z-score for a text of T tokens containing $|G|$ green tokens, with green-list fraction γ.

(c) With $\gamma = 1/2$ and $T = 100$, give the mean and standard deviation of the green count for unwatermarked text.

(d) A suspect text has 70 green tokens. Compute z. Then say what happens to the score if a user paraphrases the text before it is tested.

(4 points. Source: Ed.)

Answer 100:

(a) At each step a hash of the previous token seeds a random split of the vocabulary into a green list of fraction γ and a red list. A constant δ is added to the green logits before sampling, so green tokens become more likely without the text being forced. Detection needs only the key and the text, not the model.

(b)
$$z = \frac{|G| - \gamma T}{\sqrt{\gamma(1-\gamma)T}}. \tag{61}$$

(c) Under the null hypothesis each token is green with probability $1/2$, so the mean is 50, the variance $100 \times \frac{1}{2} \times \frac{1}{2} = 25$ and the standard deviation 5.

(d) $z = (70 - 50)/5 = 4$, four standard deviations above chance, strong evidence of the watermark. Paraphrasing replaces tokens and changes the previous-token hashes, so the green count drifts back toward 50 and z falls. The watermark tests for the exact token sequence, not for its meaning.

Reference

Every paper named in a question, in order of first appearance. The arXiv identifier is a link.

- Kingma and Ba, 2014. *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980
- Mikolov et al., 2013. *Efficient Estimation of Word Representations in Vector Space*. arXiv:1301.3781
- Bojanowski et al., 2016. *Enriching Word Vectors with Subword Information*. arXiv:1607.04606
- Mikolov et al., 2013. *Distributed Representations of Words and Phrases and their Compositionality*. arXiv:1310.4546
- Bahdanau et al., 2014. *Neural Machine Translation by Jointly Learning to Align and Translate*. arXiv:1409.0473
- Luong et al., 2015. *Effective Approaches to Attention-based Neural Machine Translation*. arXiv:1508.04025
- Beltagy et al., 2020. *Longformer: The Long-Document Transformer*. arXiv:2004.05150
- Devlin et al., 2018. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv:1810.04805
- Su et al., 2021. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. arXiv:2104.09864
- Xiong et al., 2020. *On Layer Normalization in the Transformer Architecture*. arXiv:2002.04745
- Raffel et al., 2019. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. arXiv:1910.10683
- Wei et al., 2022. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. arXiv:2201.11903
- Wang et al., 2022. *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. arXiv:2203.11171
- Zhao et al., 2021. *Calibrate Before Use: Improving Few-Shot Performance of Language Models*. arXiv:2102.09690
- Schaeffer et al., 2023. *Are Emergent Abilities of Large Language Models a Mirage?*. arXiv:2304.15004
- Rafailov et al., 2023. *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. arXiv:2305.18290
- Holtzman et al., 2019. *The Curious Case of Neural Text Degeneration*. arXiv:1904.09751
- Hu et al., 2021. *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv:2106.09685
- Shazeer, 2019. *Fast Transformer Decoding: One Write-Head is All You Need*. arXiv:1911.02150
- Ainslie et al., 2023. *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*. arXiv:2305.13245
- Leviathan et al., 2022. *Fast Inference from Transformers via Speculative Decoding*. arXiv:2211.17192
- Dao et al., 2022. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. arXiv:2205.14135

- Liu et al., 2022. *Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning*. arXiv:2205.05638
- Li and Liang, 2021. *Prefix-Tuning: Optimizing Continuous Prompts for Generation*. arXiv:2101.00190
- Rajbhandari et al., 2019. *ZeRO: Memory Optimizations Toward Training Trillion Parameter Models*. arXiv:1910.02054
- van den Oord et al., 2018. *Representation Learning with Contrastive Predictive Coding*. arXiv:1807.03748
- Karpukhin et al., 2020. *Dense Passage Retrieval for Open-Domain Question Answering*. arXiv:2004.04906
- Khattab and Zaharia, 2020. *ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT*. arXiv:2004.12832
- Santhanam et al., 2021. *ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction*. arXiv:2112.01488
- Santhanam et al., 2022. *PLAID: An Efficient Engine for Late Interaction Retrieval*. arXiv:2205.09707
- Lewis et al., 2020. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv:2005.11401
- Khattab et al., 2021. *Relevance-Guided Supervision for OpenQA with ColBERT*. arXiv:2007.00814
- Yao et al., 2022. *ReAct: Synergizing Reasoning and Acting in Language Models*. arXiv:2210.03629
- Snell et al., 2024. *Scaling LLM Test-Time Compute Optimally Can Be More Effective than Scaling Model Parameters*. arXiv:2408.03314
- Shao et al., 2024. *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models*. arXiv:2402.03300
- Khattab et al., 2023. *DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines*. arXiv:2310.03714
- Jimenez et al., 2023. *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?*. arXiv:2310.06770
- Liu et al., 2023. *Lost in the Middle: How Language Models Use Long Contexts*. arXiv:2307.03172
- Chen et al., 2021. *Evaluating Large Language Models Trained on Code*. arXiv:2107.03374
- Schick et al., 2023. *Toolformer: Language Models Can Teach Themselves to Use Tools*. arXiv:2302.04761
- Sennrich et al., 2015. *Neural Machine Translation of Rare Words with Subword Units*. arXiv:1508.07909
- Jiang et al., 2024. *Mixtral of Experts*. arXiv:2401.04088
- Katharopoulos et al., 2020. *Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention*. arXiv:2006.16236
- Gu and Dao, 2023. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. arXiv:2312.00752
- Nie et al., 2025. *Large Language Diffusion Models*. arXiv:2502.09992
- Hewitt and Liang, 2019. *Designing and Interpreting Probes with Control Tasks*. arXiv:1909.03368
- Cunningham et al., 2023. *Sparse Autoencoders Find Highly Interpretable Features in Language Models*. arXiv:2309.08600
- Gao et al., 2022. *Scaling Laws for Reward Model Overoptimization*. arXiv:2210.10760
- Hoffmann et al., 2022. *Training Compute-Optimal Large Language Models*. arXiv:2203.15556
- Kirchenbauer et al., 2023. *A Watermark for Large Language Models*. arXiv:2301.10226