CS109 Lecture 23: LLM Learning Guide

Deep Learning

Autumn 2026

Lecture 23: LLM Learning Guide

Before lecture: read the Lecture 22 slides (deep learning). Then open your favorite LLM and work through the concepts below **in order**. Each concept has a *Learn* prompt and a *Test me* prompt. Attempt the "Test me" questions yourself before asking the LLM to grade you.

How to get the most out of this:

- Tell the LLM your level: "I'm a student in Stanford's CS109 (probability for computer scientists). Explain at that level."
- After any answer, ask "why?" at least once and ask it to show each step.
- When it states a formula, ask it to define every symbol before continuing.
- Attempt practice yourself first, then say: "Here is my reasoning: _____. Where exactly am I wrong, if anywhere?"

A warning specific to today: this is the one lecture where you are asking an LLM to explain how LLMs work, and there is an enormous amount of deep learning content on the internet written at a completely different level of abstraction than CS109 uses. Insist on the CS109 framing — maximum likelihood estimation, log-likelihood, gradient ascent, the sigmoid derivative — and push back if it starts talking about optimizers, PyTorch, or transformer architectures. You want the derivation, not the ecosystem.

Concept 1: Softmax and multi-class prediction

Learn: > "Explain the softmax function $\text{softmax}(z)_k = \frac{e^{z_k}}{\sum_j e^{z_j}}$ as the generalization of the sigmoid from binary to K classes: 'sigmoid is to Bernoulli as softmax is to Categorical.' Show that the outputs are positive and sum to 1, that the function depends only on differences between the z_k, and that it never outputs exactly 0. Then prove algebraically that for $K = 2$ softmax reduces to $\sigma(z_1 - z_2)$. Finally, explain why we use e^{z_k} rather than a simpler normalization like $\sigma(z_k)/\sum_j \sigma(z_j)$ — specifically, what goes wrong with the concavity of the log-likelihood."

Test me: > "Give me several vectors z and make me compute softmax by hand, including one where all entries are equal and one where entries are far apart. Then make me prove the $K = 2$ reduction to sigmoid from scratch. Then ask me what the log-likelihood of one datapoint looks like when the label is one-hot, and why the answer simplifies to a single log term."

Concept 2: A neural network is stacked logistic regressions

Learn: > "Explain the core CS109 definition: a neural network is many logistic regression units stacked on top of each other, and deep learning is maximum likelihood estimation with neural networks. Set up

the notation carefully: input layer x, hidden layer h, output $\hat{y}$; $\theta^{(1)}_{j,k}$ is the weight from input k into hidden neuron j; $\theta^{(2)}_j$ is the weight from hidden neuron j into the output. Explain that each hidden neuron computes $h_j = \sigma\left(\sum_k \theta^{(1)}_{j,k} x_k\right)$ — an ordinary logistic regression on the inputs — and the output computes $\hat{y} = \sigma\left(\sum_j \theta^{(2)}_j h_j\right)$, an ordinary logistic regression whose *features are the hidden activations*. Then connect this to the biological neuron picture and explain what 'learning your own features' means. Finally, explain why the sigmoid nonlinearity is essential: without it the whole network collapses to a single linear model."

Test me: > "Give me a small network's weights and one datapoint, and make me do the complete forward pass by hand: every weighted sum, every sigmoid, and the final $\hat{y}$. Then make me compute the log-likelihood of that datapoint. Then ask me to count the parameters in networks of various shapes, including with and without bias terms, and ask me what would happen if I deleted the hidden-layer sigmoids."

Concept 3: The deep learning assumption and log-likelihood

Learn: > "Show me that the probability model for a neural network binary classifier is *identical* to the one for logistic regression: assume $Y \mid X = x \sim \text{Bern}(\hat{y})$, write the differentiable Bernoulli PMF, use the i.i.d. assumption to multiply across datapoints, and take the log to get >

$$LL = \sum_i y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}).$$

> Be precise about what actually changed relative to Lecture 20: not the probability model, only the function that computes $\hat{y}$. Then state the three steps we always do — make the assumption, write the log-likelihood, take partial derivatives with respect to every parameter — and explain why step 3 is necessary (we need it for gradient ascent, because there is no closed-form solution)."

Test me: > "Ask me to state the deep learning assumption and derive the log-likelihood from the Bernoulli PMF with no hints. Then quiz me: what is different between logistic regression and a neural network, what is the same, and why can't we just set the derivative to zero and solve like we did for the Poisson MLE?"

Concept 4: Backpropagation, output layer

Learn: > "Derive $\frac{\partial LL}{\partial \theta^{(2)}_j} = h_j\,[y - \hat{y}]$ for the output layer of a neural network. Use the chain rule in three pieces: $\frac{\partial LL}{\partial \hat{y}} = \frac{y - \hat{y}}{\hat{y}(1-\hat{y})}$, $\frac{\partial \hat{y}}{\partial z^{(2)}} = \hat{y}(1-\hat{y})$ (the sigmoid derivative identity), and $\frac{\partial z^{(2)}}{\partial \theta^{(2)}_j} = h_j$. Show the cancellation explicitly. Then point out that this is *exactly* the logistic regression gradient from Lecture 20 with x_j replaced by h_j, and explain what that means: the output layer is a logistic regression that cannot tell its features were computed rather than measured."

Test me: > "Make me reproduce the output-layer derivation from scratch, grading each step. Then give me a concrete tiny network and datapoint and make me compute every numeric output-layer gradient, do one gradient ascent step with a given η, recompute $\hat{y}$, and verify the log-likelihood went up. Catch me if I forget the bias term."

Concept 5: Backpropagation, hidden layer

Learn: > "Derive the hidden-layer gradient $\frac{\partial LL}{\partial \theta^{(1)}_{j,k}} = [y - \hat{y}]\,\theta^{(2)}_j\, h_j(1 - h_j)\, x_k$. First write out the full chain of dependencies from $\theta^{(1)}_{j,k}$ through $z^{(1)}_j$, h_j, $z^{(2)}$, $\hat{y}$, to LL, naming each intermediate quantity. Then give the four partial derivatives and multiply. Then read the formula back to me in English — error at the output, times how much this neuron mattered, times how sensitive this neuron is right now, times how active this input was. Explain why the algorithm is called *back*propagation, emphasizing that the error signal $y - \hat{y}$ is computed once at the output and reused for every parameter as it flows backward. Finally, note that

$h_j(1 - h_j)$ is largest when $h_j = 0.5$ and near zero when h_j is saturated near 0 or 1, and explain how that leads to the vanishing gradient problem in deep networks."

Test me: > "Give me a concrete two-input, two-hidden-neuron, one-output network with weights and a labelled datapoint, and make me compute every hidden-layer gradient numerically. Include at least one input feature equal to 0 and one hidden neuron near saturation, and then ask me to explain the size of each resulting gradient. Then make me derive the general formula from scratch with no hints and grade every step."

Concept 6: Why any of this works (and when it doesn't)

Learn: > "Compare optimization for logistic regression versus neural networks. Logistic regression has a concave log-likelihood, so gradient ascent reaches the global optimum; neural network log-likelihoods are non-concave, with many local optima and saddle points. Explain the practical consequences: dependence on random initialization, no optimality guarantee, need for a fixed seed to reproduce results. Then explain why deep learning still works empirically. Finally, tie it all back to Lecture 21: a neural network is just another entry in the model comparison table, so it still needs a train/test split, a baseline, calibration checking, and a fairness audit — and mention that the adversarial debiasing work discussed in class used neural networks for exactly that purpose."

Test me: > "Quiz me on the differences between logistic regression and neural networks along several axes: expressive power, number of parameters, concavity of the objective, interpretability of the weights, and amount of data required. Then give me a scenario and make me argue for logistic regression over a neural network, and then argue the reverse. Push back if I claim more parameters is automatically better."

Wrap-up prompt (after all six concepts)

> "Give me one multi-part CS109-style problem in which I must (1) compute a softmax and show it reduces to a sigmoid for two classes, (2) count the parameters in a described network, (3) do a complete forward pass by hand and report the log-likelihood of the datapoint, (4) derive and numerically compute the output-layer gradients, and (5) derive and numerically compute a hidden-layer gradient using the chain rule. Let me solve every part, then grade each part, tell me which concept it tested, and tell me the single concept I should review most."

Note: this is the last new machine learning content of the quarter, and it is worth pausing on what just happened. Nine weeks ago we defined probability with three axioms. Today we derived, from those axioms, the training algorithm behind essentially every modern AI system — including the one you have been prompting all quarter. The whole path was: model the world with a random variable, write the likelihood of your data, take the log, take the derivative, walk uphill.