CS109 Lecture 23 Worksheet: ANSWER KEY

Deep Learning

Autumn 2026

Lecture 23 Worksheet: Answer Key

Instructor / TA copy.

Problem 1 (Review): Calibration and Baselines

(a) Observed fraction = $36/60 = 0.60$, but the model claimed 0.80. Not calibrated, and **overconfident** — it asserts more certainty than the outcomes support.

(b) "Always predict 1" is right on every label-1 datapoint: accuracy = 0.70.

(c) Yes. A model that outputs the base rate 0.70 for every datapoint is perfectly calibrated, but thresholded at 0.5 it predicts 1 for everyone — so it *ties* the baseline. A calibrated model with mild discriminative power in the wrong places can easily do worse. Calibration constrains what the numbers mean, not how good the decisions are.

Problem 2: Softmax

(a) $\text{softmax}([7,7,7,7]) = [0.25, 0.25, 0.25, 0.25]$. Softmax depends only on the *differences* between the z_k; with all inputs equal there is no evidence favoring any class, so the result is uniform.

(b) With $z = [1,3,0]$: $e^1 = 2.718$, $e^3 = 20.086$, $e^0 = 1$, summing to 23.804.

$$\text{softmax}([1,3,0]) = [0.114,\ 0.844,\ 0.042].$$

(c) Sum: $\sum_k \frac{e^{z_k}}{\sum_j e^{z_j}} = \frac{\sum_k e^{z_k}}{\sum_j e^{z_j}} = 1$. Each term is positive (exponentials are always > 0) and each numerator is strictly less than the sum, so every output lies in $(0,1)$. The largest z_k gets the largest probability because e^z is strictly increasing. It never outputs exactly 0 because $e^z > 0$ for every finite z — which is a feature, not a bug: it means the model never assigns impossibility to a class, and $\log \hat{y}_k$ is always defined.

(d) Divide numerator and denominator by e^{z_1}:

$$\frac{e^{z_1}}{e^{z_1} + e^{z_2}} = \frac{1}{1 + e^{z_2 - z_1}} = \frac{1}{1 + e^{-(z_1 - z_2)}} = \sigma(z_1 - z_2).$$

So sigmoid is the two-class special case: "sigmoid is to Bernoulli as softmax is to Categorical."

(e) The normalized-logistic version produces a valid distribution, but the resulting log-likelihood is **not concave**, so gradient ascent can get stuck in local optima and training is no longer reliable. Softmax's log-likelihood is concave in the z_k (and the exponential turns into a clean $\log \sum e^{z_j}$ term under the log), which is exactly the property we leaned on in Lecture 20. Convexity is not an aesthetic preference — it is what makes the optimization trustworthy.

Problem 3: Counting Parameters

(a) Every input connects to every hidden neuron: $40 \times 20 = \boxed{800}$.

(b) Every hidden neuron connects to the single output: $\boxed{20}$.

(c) $800 + 20 = \boxed{820}$.

(d) $(40 \times 20) + (20 \times 20) + 20 = 800 + 400 + 20 = \boxed{1220}$.

(e) One bias per neuron: 20 hidden neurons plus 1 output neuron = 21 extra, for 841 total. Each bias is exactly the θ_0 intercept from Lecture 20 — the weight on a constant $x_0 = 1$ input — and it lets a neuron shift its activation threshold away from $z = 0$.

Scale note worth mentioning at the board: this toy network has 820 parameters. A modern large language model has on the order of 10^{11}. The derivation on this worksheet is, mechanically, the same one running there.

Problem 4: A Forward Pass by Hand

(a) With $x = [1, 1, 0]$ (bias prepended):

$$z_1 = 0(1) + 2(1) + (-2)(0) = 2 \implies h_1 = \sigma(2) = 0.8808.$$

$$z_2 = -1(1) + 1(1) + 3(0) = 0 \implies h_2 = \sigma(0) = 0.5.$$

(b) With $h = [1, 0.8808, 0.5]$:

$$z^{(2)} = -1(1) + 2(0.8808) + 1(0.5) = 1.2616 \implies \hat{y} = \sigma(1.2616) = 0.7793.$$

(c) $y = 1$, so $LL = \log(0.7793) \approx -0.2494$.

(d) Three logistic regressions were evaluated:

1. $h_1 = \sigma(\theta_1^{(1)} \cdot x)$ — input vector $[1, 1, 0]$.
2. $h_2 = \sigma(\theta_2^{(1)} \cdot x)$ — the *same* input vector $[1, 1, 0]$, different weights.
3. $\hat{y} = \sigma(\theta^{(2)} \cdot h)$ — input vector $[1, 0.8808, 0.5]$, i.e. the outputs of the first two.

The only new idea in a neural network is that the *inputs of one logistic regression are the outputs of others.* Everything else is Lecture 20.

Problem 5: Backpropagation, Output Layer

(a) $\frac{\partial z^{(2)}}{\partial \theta_j^{(2)}} = h_j$, so by the chain rule

$$\frac{\partial LL}{\partial \theta_j^{(2)}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial \theta_j^{(2)}} = \frac{y - \hat{y}}{\hat{y}(1 - \hat{y})} \cdot \hat{y}(1 - \hat{y}) \cdot h_j = h_j \left[y - \hat{y} \right].$$

The $\hat{y}(1 - \hat{y})$ cancels, exactly as in Lecture 20.

(b) Nothing changed except that x_j became h_j. The output layer is *literally* a logistic regression whose features are the hidden activations. It has no idea that its inputs were themselves computed — it just sees a feature vector. This is why deep learning is often described as "learning your own features": the hidden layers invent the h_j that make the final logistic regression's job easy.

(c) $y - \hat{y} = 1 - 0.7793 = 0.2207$, and $h = [1, 0.8808, 0.5]$:

$$\frac{\partial LL}{\partial \theta_0^{(2)}} = 0.2207, \qquad \frac{\partial LL}{\partial \theta_1^{(2)}} = 0.1944, \qquad \frac{\partial LL}{\partial \theta_2^{(2)}} = 0.1103.$$

Note that h_1 was more active than h_2, so it receives a proportionally larger share of the update.

(d) $\theta^{(2)} \leftarrow [-0.7793,\ 2.1944,\ 1.1103]$. Recomputing:

$$z^{(2)} = -0.7793 + 2.1944(0.8808) + 1.1103(0.5) = 1.7087 \implies \hat{y} = \sigma(1.7087) = 0.8467.$$

LL rose from -0.2494 to $\log(0.8467) \approx -0.1664$. Improved, as gradient ascent guarantees for a small enough step.

Problem 6: Backpropagation, Hidden Layer

(a) The chain of influence is

$$\theta_{j,k}^{(1)} \longrightarrow z_j^{(1)} \longrightarrow h_j \longrightarrow z^{(2)} \longrightarrow \hat{y} \longrightarrow LL,$$

which we group as

$$\frac{\partial LL}{\partial \theta_{j,k}^{(1)}} = \underbrace{\frac{\partial LL}{\partial z^{(2)}}}_{y-\hat{y}} \cdot \underbrace{\frac{\partial z^{(2)}}{\partial h_j}}_{\theta_j^{(2)}} \cdot \underbrace{\frac{\partial h_j}{\partial z_j^{(1)}}}_{h_j(1-h_j)} \cdot \underbrace{\frac{\partial z_j^{(1)}}{\partial \theta_{j,k}^{(1)}}}_{x_k}.$$

(b) Multiplying the four pieces:

$$\frac{\partial LL}{\partial \theta_{j,k}^{(1)}} = [y - \hat{y}]\, \theta_j^{(2)}\, h_j(1 - h_j)\, x_k.$$

Notice the first factor, $y - \hat{y}$, is *reused* from the output-layer calculation. That reuse is the entire efficiency argument for backpropagation: you compute the error signal once at the output and pass it backward, rather than recomputing a fresh derivative from scratch for each of the 820 (or 10^{11}) parameters.

(c) With $y - \hat{y} = 0.2207$, $h_1 = 0.8808$, $h_1(1 - h_1) = 0.1050$, $h_2 = 0.5$, $h_2(1 - h_2) = 0.25$, $x_1 = 1$:

$$\frac{\partial LL}{\partial \theta_{1,1}^{(1)}} = 0.2207 \times 2 \times 0.1050 \times 1 \approx \boxed{0.0463},$$

$$\frac{\partial LL}{\partial \theta_{2,1}^{(1)}} = 0.2207 \times 1 \times 0.25 \times 1 \approx \boxed{0.0552}.$$

Worth pointing out: h_2 has the *smaller* outgoing weight ($\theta_2^{(2)} = 1$ vs. 2) yet gets the *larger* update, because $h_2 = 0.5$ sits at the steepest point of the sigmoid where $h(1 - h)$ is maximized. $h_1 = 0.88$ is already partly saturated and responds sluggishly. This is the seed of the vanishing-gradient problem.

(d) $x_2 = 0$, so

$$\frac{\partial LL}{\partial \theta_{1,2}^{(1)}} = 0.$$

That input was off for this datapoint, so the weight attached to it had no effect on the prediction and receives neither credit nor blame. Same logic as $x_j = 0$ in Lecture 20.

(e) English reading: *(error at the output) × (how much this neuron mattered to the output) × (how sensitive this neuron is right now) × (how active this input was).*

It is called *back*propagation because the error signal $y - \hat{y}$ is computed at the output and then propagated backward through the network, getting multiplied by one local factor per layer as it goes. The forward pass carries activations left to right; the backward pass carries responsibility right to left.

Problem 7: Deep Learning Is Maximum Likelihood Estimation

(a) "Deep learning is **maximum likelihood estimation** with neural networks." The three steps, unchanged since Lecture 19: (1) make the modeling assumption, (2) calculate the log-likelihood of all the data, (3) take the partial derivative of the log-likelihood with respect to each parameter (and then run gradient ascent).

(b) Nothing about the *probability* model changed — both assume $Y \mid X = x \sim \text{Bern}(\hat{y})$ and both maximize the same Bernoulli log-likelihood. What changed is the *function* computing $\hat{y}$: logistic regression uses $\sigma(\theta^T x)$, a single linear-then-squash step, while the network composes many of them. Richer function, identical estimation framework.

(c) Because there is no closed form: we cannot solve $\frac{\partial LL}{\partial \theta} = 0$ for hundreds of coupled nonlinear equations. So we use an optimization technique — gradient ascent — and gradient ascent needs the partial derivatives. If we *could* solve it in closed form we would simply do that, exactly as we did for the Bernoulli, Poisson, and exponential MLEs.

(d) **No, it is not concave.** Neural network log-likelihoods have many local optima (and saddle points). Consequences: the result depends on the random initialization, you may need to restart or tune, you have no guarantee of finding the global optimum, and reported results are not exactly reproducible without fixing a seed. In practice good local optima are usually good enough — which is a lucky empirical fact, not a theorem.

(e) The whole network would collapse into a single linear model. A composition of linear maps is linear: $\theta^{(2)}\left(\theta^{(1)}x\right) = \left(\theta^{(2)}\theta^{(1)}\right)x = \theta' x$. You would have an ordinary logistic regression with extra steps, and no ability to represent a nonlinear decision boundary. The nonlinearity in the hidden layer is what buys the expressive power.

Challenge Problem: The Multi-Class Log-Likelihood

(a) The categorical PMF written in differentiable form is $P(Y = y) = \prod_{k=1}^{K} \hat{y}_k^{\,y_k}$, so

$$LL = \sum_{k=1}^{K} y_k \log \hat{y}_k.$$

Since y is one-hot, every term vanishes except the one for the true class c:

$$LL = \log \hat{y}_c.$$

(In machine learning this is called the cross-entropy loss; note the direct line back to the Information Theory lecture.)

(b) With $K = 2$, write $y = [y_1, y_2] = [y, 1 - y]$ and $\hat{y} = [\hat{y}, 1 - \hat{y}]$. Then

$$LL = y \log \hat{y} + (1 - y) \log(1 - \hat{y}),$$

the Bernoulli log-likelihood exactly.

(c) For the correct class c, $LL = \log \hat{y}_c$ and $y_c = 1$, so

$$\frac{\partial LL}{\partial z_c} = \frac{1}{\hat{y}_c} \cdot \frac{\partial \hat{y}_c}{\partial z_c} = \frac{1}{\hat{y}_c} \cdot \hat{y}_c(1 - \hat{y}_c) = 1 - \hat{y}_c = y_c - \hat{y}_c.$$

(For an incorrect class $k \neq c$ one uses $\frac{\partial \hat{y}_c}{\partial z_k} = -\hat{y}_c \hat{y}_k$, giving $-\hat{y}_k = y_k - \hat{y}_k$ since $y_k = 0$. Same form.)

(d) It is not a coincidence. Sigmoid and softmax are the *canonical link functions* for the Bernoulli and Categorical distributions — members of the exponential family — and for any such pairing the derivative of the log-likelihood with respect to the natural parameter is exactly (observed − expected). The $\hat{y}(1 - \hat{y})$ that cancels so satisfyingly in every derivation on this worksheet is the same cancellation showing up in different costumes. Students do not need this vocabulary for CS109; the takeaway is that the clean form is structural, not lucky, which is why the same three-line update rule trains a two-parameter logistic regression and a hundred-billion-parameter language model.