CS109 Lecture 23 Worksheet: Deep Learning

Softmax, Neural Networks, Forward Passes, and Backpropagation

Autumn 2026

Lecture 23 Worksheet: Deep Learning

Topics: softmax and multi-class classification, neural networks as stacked logistic regressions, counting parameters, the forward pass, the deep learning log-likelihood, and deriving backpropagation with the chain rule.

How to use this sheet. Work in small groups. A TA and the instructor will circulate to help. We will go over selected solutions on the board. Today's payoff is that *you can derive backpropagation yourself* — it is the chain rule and the sigmoid identity, nothing more.

Useful facts. $\sigma(z) = \dfrac{1}{1+e^{-z}}$, $\quad \sigma'(z) = \sigma(z)[1-\sigma(z)]$, $\quad \sigma(0) = 0.5$, $\quad \sigma(2) = 0.8808$, $\quad \sigma(1.2616) = 0.7793$.

Notation (matching the slides). Layer x is the input, layer h is the hidden layer, and $\hat{y}$ is the output. $\theta^{(1)}_{j,k}$ is the weight from input x_k into hidden neuron h_j; $\theta^{(2)}_j$ is the weight from hidden neuron h_j into the output.

Problem 1 (Review of Lecture 21): Calibration and Baselines

A classifier is evaluated on 300 test datapoints. Of the 60 datapoints where it output a probability near 0.8, exactly 36 turned out to have label 1. The dataset is 70% label 1 overall.

(a) Is the model calibrated in that bucket? Overconfident or underconfident?

(b) What accuracy does the "always predict 1" baseline achieve on this dataset?

(c) In one sentence: could a model be perfectly calibrated and still lose to that baseline on accuracy?

Problem 2: Softmax

Logistic regression outputs a single probability for a binary label. For K classes we use the **softmax**:

$$\text{softmax}(z)_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}.$$

(a) Compute softmax$([7, 7, 7, 7])$. Explain the answer in one sentence.

(b) Compute softmax$([1, 3, 0])$ to three decimal places.

(c) Verify that the outputs of softmax always sum to 1 and are always in $(0, 1)$. Which class gets the largest probability, and why does softmax never output an exact 0?

(d) Show that for $K = 2$, softmax reduces to the sigmoid. Specifically, show that

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

(e) A classmate proposes the simpler "normalized logistic" $\frac{\sigma(z_k)}{\sum_j \sigma(z_j)}$ instead. It also produces numbers in $(0, 1)$ that sum to 1. Why did we not use it? *(Hint: think about what property of the log-likelihood we relied on for gradient ascent in Lecture 20.)*

Problem 3: Counting Parameters

Consider a neural network with an input layer of size $|x| = 40$, one hidden layer of size $|h| = 20$, and a single output neuron. Ignore bias terms for this problem.

(a) How many parameters are in $\theta^{(1)}$ (input layer $\rightarrow$ hidden layer)?

(b) How many parameters are in $\theta^{(2)}$ (hidden layer $\rightarrow$ output)?

(c) How many parameters in total?

(d) Now add a second hidden layer of size 20 between the first hidden layer and the output. How many total parameters?

(e) If we included a bias term for each neuron, how many parameters would the original network have? What does each bias term correspond to in the logistic regression language of Lecture 20?

Problem 4: A Forward Pass by Hand

Here is a tiny network with 2 inputs, 2 hidden neurons, and 1 output. Every neuron is a logistic regression unit: it takes a weighted sum of its inputs (with $x_0 = 1$ and $h_0 = 1$ prepended as bias terms) and applies a sigmoid.

$$\theta_1^{(1)} = [0,\ 2,\ -2] \quad (\text{into } h_1), \qquad \theta_2^{(1)} = [-1,\ 1,\ 3] \quad (\text{into } h_2), \qquad \theta^{(2)} = [-1,\ 2,\ 1].$$

The datapoint is $x = (x_1, x_2) = (1, 0)$ with true label $y = 1$.

(a) Compute the weighted sum into h_1, then $h_1 = \sigma(\cdot)$. Do the same for h_2.

(b) Compute the weighted sum into the output neuron, then $\hat{y}$.

(c) What is the log-likelihood of this single datapoint under the current parameters?

(d) The network is a stack of logistic regressions. Identify precisely which three logistic regressions were evaluated in parts (a) and (b), and what each one's "input vector" was.

Problem 5: Backpropagation, Output Layer

Keep the network and datapoint from Problem 4. Because $Y \mid X = x \sim \text{Bern}(\hat{y})$ — the *same* assumption as logistic regression — the log-likelihood is

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

(a) Let $z^{(2)} = \sum_j \theta_j^{(2)} h_j$ be the weighted sum into the output. Using the chain rule with $\frac{\partial LL}{\partial \hat{y}} = \frac{y - \hat{y}}{\hat{y}(1-\hat{y})}$ and $\frac{\partial \hat{y}}{\partial z^{(2)}} = \hat{y}(1 - \hat{y})$, show that

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

(b) Compare this to the logistic regression gradient from Lecture 20. What changed? What does that tell you about how the output layer "sees" the rest of the network?

(c) Compute all three numeric values $\frac{\partial LL}{\partial \theta_0^{(2)}}$, $\frac{\partial LL}{\partial \theta_1^{(2)}}$, $\frac{\partial LL}{\partial \theta_2^{(2)}}$ for the datapoint in Problem 4.

(d) With step size $\eta = 1$, perform one gradient ascent update on $\theta^{(2)}$ only, then recompute $\hat{y}$. Did the log-likelihood improve?

Problem 6: Backpropagation, Hidden Layer

This is the heart of deep learning. We want $\frac{\partial LL}{\partial \theta_{j,k}^{(1)}}$ — the derivative with respect to a weight that is buried one layer deep.

(a) Write the chain of dependencies: how does $\theta_{j,k}^{(1)}$ influence LL? Write the chain as a product of four partial derivatives, naming each intermediate quantity.

(b) Using $\frac{\partial LL}{\partial z^{(2)}} = y - \hat{y}$ from Problem 5, $\frac{\partial z^{(2)}}{\partial h_j} = \theta_j^{(2)}$, and $\frac{\partial h_j}{\partial \theta_{j,k}^{(1)}} = h_j(1 - h_j)x_k$, assemble the result:

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

(c) Compute $\frac{\partial LL}{\partial \theta_{1,1}^{(1)}}$ and $\frac{\partial LL}{\partial \theta_{2,1}^{(1)}}$ numerically for the datapoint in Problem 4.

(d) Compute $\frac{\partial LL}{\partial \theta_{1,2}^{(1)}}$. Explain the answer in one sentence.

(e) Read the formula out loud in English: the error at the output, times how much this neuron mattered to the output, times how sensitive this neuron is right now, times how active this input was. **You have just derived backpropagation.** In one sentence, why is it called "back" propagation?

Problem 7: Deep Learning Is Maximum Likelihood Estimation

Answer each in one or two sentences.

(a) Complete the sentence from lecture: "Deep learning is _____ with neural networks." Then state the three things we always have to do, in order.

(b) The log-likelihood in Problem 5 is *identical* to the logistic regression log-likelihood from Lecture 20. What, precisely, is different between the two models?

(c) Why do we need the partial derivatives at all? What would we do if we could solve $\frac{\partial LL}{\partial \theta} = 0$ in closed form?

(d) The logistic regression log-likelihood is concave, so gradient ascent finds the global optimum. Is the neural network log-likelihood concave in θ? What practical consequence does your answer have?

(e) A network with a hidden layer can separate data that is not linearly separable, unlike logistic regression. What would happen if we removed the sigmoids from the hidden layer and just passed the weighted sums through?

Challenge Problem (optional): The Multi-Class Log-Likelihood

Suppose the output layer is a softmax over K classes rather than a single sigmoid, so the network outputs $\hat{y}_1, \ldots, \hat{y}_K$ with $\sum_k \hat{y}_k = 1$. The true label is one-hot: $y_k = 1$ for the correct class and 0 otherwise.

(a) Write the log-likelihood of a single datapoint. Simplify it, given that y is one-hot.

(b) Show that when $K = 2$ this reduces exactly to the Bernoulli log-likelihood we have been using.

(c) Let z_k be the weighted sum feeding into softmax output k. It turns out that $\frac{\partial LL}{\partial z_k} = y_k - \hat{y}_k$ — the same suspiciously clean "error" form we keep getting. Verify this for the case \$k = \$ the correct class, using $\frac{\partial \hat{y}_k}{\partial z_k} = \hat{y}_k(1 - \hat{y}_k)$.

(d) Speculate: why does "error $\times$ input" keep falling out of these derivations, whether the model is a Bernoulli, a categorical, or a 12-layer network?