Brown University **Alexander Khosrowshahi**
CSCI1470 **Beras: Conceptual**
Prof. Ewing **February 18, 2025**

---

1 Conceptual Questions

1. The update rule for weights in a single-layer, multi-class neural network with cross-entropy loss is defined as follows:

   Let w_{ij} be the weight associating the ith input feature x_i with the jth class. Let c be the index of the correct class for a given input (the *label*). The loss and its derivatives are then:

   $$L = -\log(P_c), \quad \frac{\partial L}{\partial w_{ij}} = \begin{cases} (P_j - 1)x_i, & j = c \\ P_j x_i, & j \neq c \end{cases}$$

   We use these partials with a learning rate α to descend along the gradient:

   $$w_{ij} = w_{ij} - \alpha \frac{\partial L}{\partial w_{ij}}$$

   Derive the above rules from the original definition of cross-entropy loss.

   Solution: We begin by expanding the definition of cross-entropy loss considering a situation where $j \neq c$.

   Cross-entropy is defined as $L = -\sum_j y_j \log(P_j)$, where y_j is the true probability of class j and P_j is the predicted probability of class j.

   In our case, y is a one-hot vector with $y_c = 1$ and $y_j = 0$ for $j \neq c$. Thus, the loss simplifies to $L = -\log(P_c)$.

   Assuming we use softmax as our activation function, the logit for class j is $z_j = \mathbf{w}_j \cdot \mathbf{x}$. The probability of class j is then

   $$P_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

   Now we can take the derivative with respect to the logit

   $$\frac{\partial L}{\partial z_j} = \frac{\partial L}{\partial P_c} \frac{\partial P_c}{\partial z_j}$$

   $$\frac{\partial L}{\partial z_j} = -\frac{1}{P_c} \frac{\partial P_c}{\partial z_j}$$

   $$\frac{\partial L}{\partial z_j} = -\frac{1}{P_c} \left(\frac{\partial}{\partial z_j} \left(\frac{e^{z_c}}{\sum_k e^{z_k}} \right) \right)$$

Which we now split into the cases where $j = c$ and $j \neq c$.

When $j = c$, we have

$$\frac{\partial P_c}{\partial z_c} = \frac{e^{z_c} \sum_k e^{z_k} - e^{z_c} e^{z_c}}{\left(\sum_k e^{z_k}\right)^2} = P_c(1 - P_c)$$

and when $j \neq c$, we have

$$\frac{\partial P_c}{\partial z_j} = \frac{-e^{z_c} e^{z_j}}{\left(\sum_k e^{z_k}\right)^2} = -P_c P_j$$

Giving us our original rules for the derivative of the loss with respect to the logit. By definition, we can use these to descend the gradient as stated.

2. In classification problems, we assign a likelihood probability to each class and use a loss function that outputs a loss based on this probability. Can you use MSE loss for classification tasks? Why or why not? Why is cross-entropy loss most commonly used for classification? (3-5 sentences)

   Hint: Think about how each loss function is shaped, the inputs they take, and their range.

3. Gradient Descent

(a) What is a gradient? How is it different from a partial derivative? How do they relate? (*2-4 sentences*)

> **Solution:** The gradient is a vector that describes the direction of steepest ascent of a function. It is composed of the partial derivatives of the function with respect to each of its variables. The partial derivative of a function is the derivative of the function with respect to one of its variables, holding all other variables constant. The gradient is a generalization of tderivative as a vector-valued function of the partial derivatives.

(b) Consider the formula for updating our weights:

$$\Delta w = -\alpha \frac{\partial L}{\partial w}$$

Why do we negate this quantity? What purpose does this serve? (*2-4 sentences*)

> **Solution:** Our goal with gradient descent is to *minimize* the loss function. The gradient points in the direction of steepest ascent, so we negate it to move in the direction of steepest *descent*. This allows us to iteratively update our weights to minimize the loss function.

(c) During gradient descent, we calculate the partial derivative of loss L with respect to each weight w_{ij}. Why must we do this for every weight? Why can't we do this for some weights? (*1-3 sentences*)

> **Solution:** We reasonably *could* compute the partial derivative of the loss with respect to a subset of the weights, however each weight contributes to each output. If we want to minimize the loss function, we must consider the effect of each weight on the output. Thus, we must compute the partial derivative of the loss with respect to each weight.

(d) In practice, most operations during gradient descent are vectorized. Why do we do this? Why might this make it beneficial to train the model on a GPU? (*1-2 sentences*)

> **Solution:** To put it simply, we vectorize our operations to correspond to the structure of such equations—vectors. Given the efficacy of GPUs of working on many parallel operations at once, it makes sense to parallelize as much as possible to speed up training.

(e) Consider the following plot of a loss function for some neural network: Where should gradient descent end up on the graph? How many weights does this model have? If our model starts training at $(-2, 2, 0)$, will the loss function ever reach the absolute minimum? Why? Assume the loss function at this point is *perfectly flat*. (*3-5 sentences*)

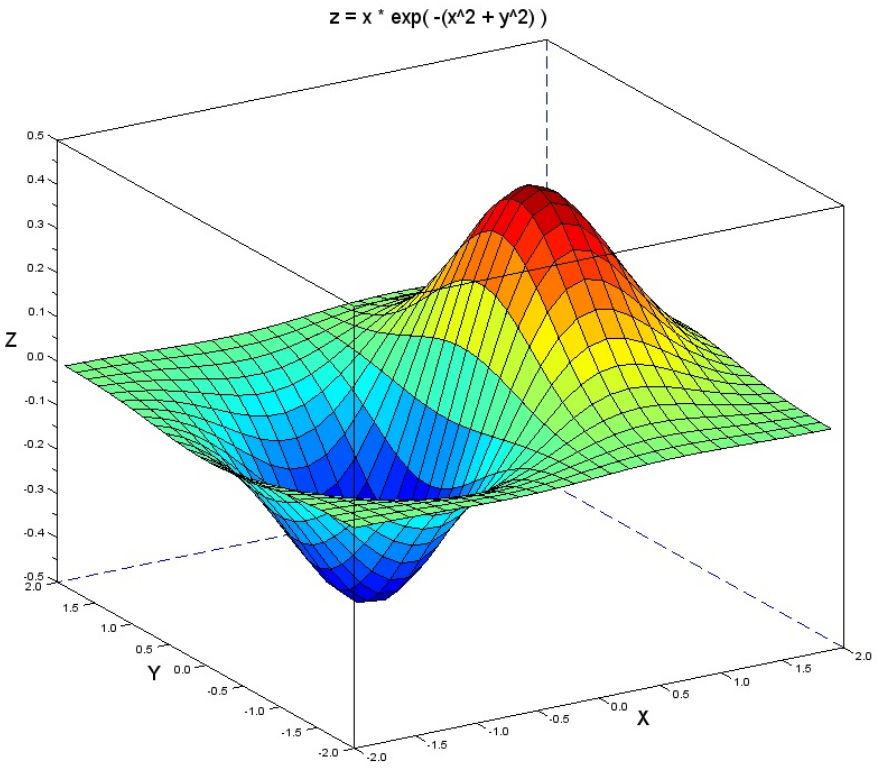

Figure 1: Loss Manifold Visualization

4. We have previously worked on single-layer linear regression using one linear function:

$$\mathbf{x} \mapsto \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$

mapping from $\mathbb{R}^s$ to $\mathbb{R}^t$. For many real-world scenarios, we actually need multiple layers to model more complex relationships.

(a) Calculate the result after we stack another linear function

$$\mathbf{x} \mapsto \mathbf{W}_2\mathbf{x} + \mathbf{b}_2$$

mapping from $\mathbb{R}^t$ to $\mathbb{R}^u$ right after the first one.

(b) What is the shape of $\mathbf{W}_1, \mathbf{b}_1$ and $\mathbf{W}_2, \mathbf{b}_2$? Explain your reasoning.

(c) Does the composition of the two linear functions offer an improvement over a single linear function? Explain your answer (*2-4 sentences*).