2.1 Lesson: Multi-Class Classification with Neural Networks

Multi-Class Classification with Neural Networks

Last week, we saw that by using a **sigmoid** activation function in the output layer, we can perform binary or multi-label classification tasks with a neural network. However, a very common task involves classifying a sample into one of a number of different classes.

Example:
Recognizing handwritten digits from the MNIST dataset (https://www.kaggle.com/datasets/hojjatk/mnist-dataset) is often considered the "Hello World" of deep learning. The input consists of 28x28 pixel black-and-white images of digits, and the output is a label from 0 to 9.

This form of classification task requires a generalization of the sigmoid function.

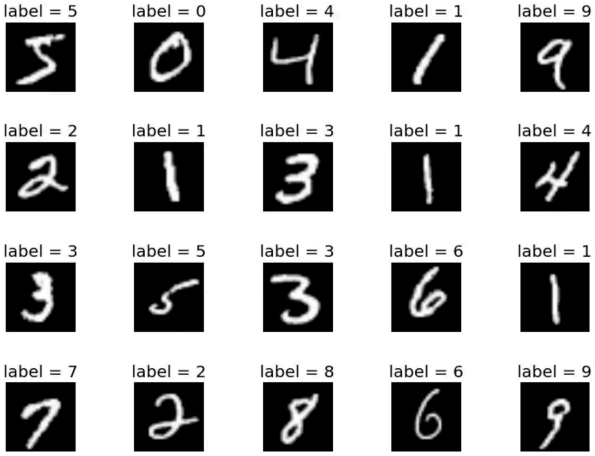

Figure 1: *MNIST Dataset*

The Softmax Function

Recall that the sigmoid turns any floating-point number into a probability between 0 and 1:

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^{z}}{e^{z}+1}$$

The **softmax** function extends the sigmoid to cases where there are $k > 1$ possible classes. It produces a probability distribution over these classes — that is, k probabilities that sum to 1.0:

$$\text{softmax}(\mathbf{z}) = \left[\frac{e^{z_1}}{\sum_{i=1}^{k} e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^{k} e^{z_i}}, \ldots, \frac{e^{z_k}}{\sum_{i=1}^{k} e^{z_i}} \right]$$

The predicted class is simply the one with the highest softmax probability.

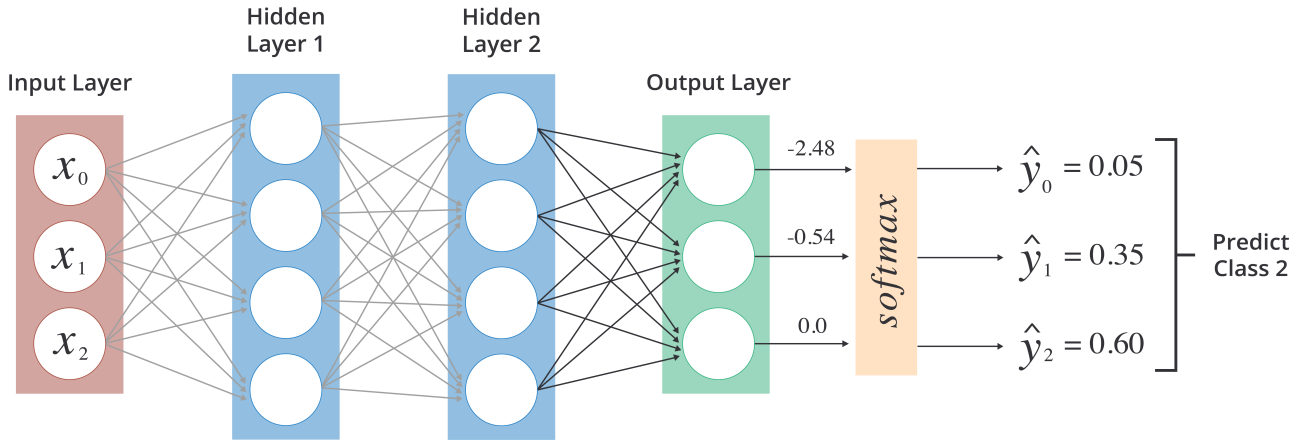

Figure 2: *A Multi-Class Classification Network*

Loss Function for Multi-Class Classification

In this form of classification, each input is assigned to exactly one class out of C classes. While the loss function is based on **cross-entropy**, how it's computed depends on how we represent the ground truth labels. There are two possibilities.

Option 1: One-Hot Encoded Labels
Just as with multi-label classification, we could encode the target as a one-hot vector of 0's and 1's. For example, the ground truth for the diagram above might be:

$$y^{(i)} = [\, y_0^{(i)}, y_1^{(i)}, y_2^{(i)} \,] = [\, 0, 0, 1 \,].$$

In this case, the **categorical cross-entropy** loss is computed as:

$$L = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{C} y_j^{(i)} \log(\hat{y}_j^{(i)})$$

where:

- $y_j^{(i)} \in 0,1$: 1 if class j is correct for sample i, 0 otherwise.
- $\hat{y}_j^{(i)} \in [0..1]$: softmax predicted probability for class j.

Notice how this formula generalizes the binary cross-entropy loss formula, where $y^{(i)}$ is either 0 or 1 and selects the appropriate part of the sum, to the one-hot vector, where exactly one of the $y_j^{(i)}$ = 1.

Option 2: Integer Class Labels

Instead of one-hot encoding, we can simply store the true class as an **integer** in the range $[0,\ldots,C-1]$. The ground truth for the above diagram would simply be 2. This approach is more efficient, widely used in practice, and is the convention in Keras/TensorFlow.

Now the loss, called Sparse Cross-Entropy Loss, simplifies to:

$$L = -\frac{1}{n}\sum_{i=1}^{n} \log\left(\hat{y}_{y^{(i)}}^{(i)}\right)$$

where:

- $y^{(i)}$ is the correct class label for sample i, used here as an index selecting the appropriate prediction.
- $\hat{y}_{y^{(i)}}^{(i)}$ is the predicted probability (from softmax) for that class.

In this case, the subscript $y^{(i)}$ indicates which class the prediction must match and ignores all the other labels.