1 Multiclass Classification

In class, we've spent a lot of time discussing *binary* classification tasks, where our labels $y^{(n)}$ can take on one of two possible values (e.g., yes or no, good or bad, pass or fail, etc...). However, many interesting real-world tasks involve more than two possible label values (e.g., classifying handwritten digits as one of "0", "1", "2", ..., "9"). This document will walk you through how we can convert some of the methods we've seen in class to handle these kinds of multiclass classification tasks.

1.1 Multinomial Logistic Regression

Multinomial logistic regression, also known as softmax regression or multiclass logistic regression, is a generalization of binary logistic regression. Suppose we have a dataset $\mathcal{D}$, such that

$$\mathcal{D} = \left\{ \left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(N)}, y^{(N)}\right) \right\} \text{ where } \mathbf{x}^{(n)} \in \mathbb{R}^D, y^{(n)} \in \{1, \ldots, K\} \text{ for } n = 1, \ldots, N$$

Here, N is the number of training examples, D is the number of features, and K is the number of possible classes, which in this setting we will assume to be greater than two. In order to perform multiclass classification, we will manipulate the output of the linear model such that we obtain the probability that $y^{(n)}$ belongs to each of the K classes. This is possible through the softmax$(\cdot)$ function,

$$p\left(Y = y^{(n)} \mid \mathbf{x}^{(n)}, \boldsymbol{\Theta}\right) = \frac{\exp\left(\boldsymbol{\Theta}_{y^{(n)}} \mathbf{x}^{(n)}\right)}{\sum_{k=1}^{K} \exp\left(\boldsymbol{\Theta}_k \mathbf{x}^{(n)}\right)} = \text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_{y^{(n)}}.$$

where $\boldsymbol{\Theta}$ is a parameter matrix of size $K \times (D+1)$. Each row of $\boldsymbol{\Theta}$ corresponds to one of the K different classes: $\boldsymbol{\Theta}_k$ denotes the kth **row** of $\boldsymbol{\Theta}$, which is the parameter vector for the kth class.

Crucially, for each data point, this model outputs a K-length vector. However, the value of $y^{(n)}$ itself is an integer between 1 and K, inclusive. As such, to define a loss function for this setting, we need to encode $y^{(n)}$ a way that allows us to calculate the differences between the predicted probability that $y^{(n)}$ takes a certain value, and the ground truth.

1.1.1 One-hot Encoding

A one-hot encoding is a vector representation of a one dimensional integer defined as such: a vector $\mathbf{c}$ of length K is a one-hot encoding of integer $k \iff |\mathbf{c}| = K$ and for all $j \neq n$, $\mathbf{c}_j = 0$ and $\mathbf{c}_k = 1$. For example, if $K = 5$ and $y^{(n)} = 2$, then a one-hot encoding of $y^{(n)}$ is:

$$[0, 1, 0, 0, 0]$$

In multinomial logistic regression, we form the matrix $\mathbf{T}$ where the nth column of $\mathbf{T}$ is the ground truth one-hot encoding of label $y^{(n)}$.

1.1.2 Cross-entropy Loss

So what's the objective function and how do we optimize it? Like in binary logistic regression, we wish to maximize the conditional log-likelihood of $y^{(n)}$ given some inputs $\mathbf{x}^{(n)}$ and parameters $\mathbf{\Theta}$. The difference with multinomial logistic regression is that we wish to generalize the binary logistic regression conditional log-likelihood formula to the case where we have multiple labels. This is done with the cross-entropy loss:

$$J^{(n)}(\mathbf{\Theta}) = -\sum_{k=1}^{K} \mathbf{T}_{kn} \log p(\mathbf{T}_{kn} = 1 \mid \mathbf{x}^{(n)}, \mathbf{\Theta})$$

where $\mathbf{T}_{kn}$ is the kth element of the nth column of $\mathbf{T}$. This entry corresponds to the whether or not the nth data point has label k: $\mathbf{T}_{kn} = 1$ if $y^{(n)} = k$ and $\mathbf{T}_{kn} = 0$ otherwise.

1.1.3 Connection to Binary Classification

Notice that $J^{(n)}(\mathbf{\Theta})$ is a generalization of binary logistic regression's negative conditional likelihood objective. To see this, let $K = 2$ and suppose the labels in binary logistic regression are one-hot encoded. Then,

$$J^{(n)}(\mathbf{\Theta}) = -\mathbf{T}_{0n} \log p(\mathbf{T}_{0n} = 1 \mid \mathbf{x}^{(n)}, \mathbf{\Theta}) - \mathbf{T}_{1n} \log p(\mathbf{T}_{1n} = 1 \mid \mathbf{x}^{(n)}, \mathbf{\Theta}).$$

More specifically, we know that $\mathbf{T}_{0n} = 1$ when $y^{(n)} = 0$, and $\mathbf{T}_{1n} = 1$ when $y^{(n)} = 1$. Thus, we can make some clever use of notation and express $\mathbf{T}_{kn}$ in this setting as $\mathbf{T}_{1n} = y^{(n)}$ and $\mathbf{T}_{0n} = 1 - y^{(n)}$. If we plug these expressions into $J^{(n)}(\mathbf{\Theta})$ we get,

$$J^{(n)}(\mathbf{\Theta}) = -y^{(n)} \log p(y^{(n)} = 1 \mid \mathbf{x}^{(n)}, \mathbf{\Theta}) - (1 - y^{(n)}) \log p((1 - y^{(n)}) = 1 \mid \mathbf{x}^{(n)}, \mathbf{\Theta}),$$

which is the binary logistic regression objective.

1.1.4 Computing Gradients

All that is left now is to take the gradient of $J^{(n)}(\mathbf{\Theta})$ (in the general case version) to derive the stochastic gradient descent update,

$$\mathbf{\Theta}_j = \mathbf{\Theta}_j - \frac{\partial J^{(n)}(\mathbf{\Theta})}{\partial \mathbf{\Theta}_j}.$$

Note that we are taking a gradient with respect to the jth *row* of $\mathbf{\Theta}$. The reason for this will become clear with the derivation.

Let's use the general-case formula for $J^{(n)}(\boldsymbol{\Theta})$ with the softmax equation substituted in for $p(\mathbf{T}_{kn} = 1 \mid \mathbf{x}^{(n)}, \boldsymbol{\Theta})$,

$$J^{(n)}(\boldsymbol{\Theta}) = -\sum_{k=1}^{K} \mathbf{T}_{kn} \log \ \text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k.$$

Then

$$\begin{aligned}
\frac{\partial J^{(n)}(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}_j} &= -\frac{\partial}{\partial \boldsymbol{\Theta}_j} \sum_{k=1}^{K} \mathbf{T}_{kn} \log \ \text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k \\
&= -\sum_{k=1}^{K} \mathbf{T}_{kn} \frac{\partial}{\partial \boldsymbol{\Theta}_j} \left(\log \ \text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k \right) \\
&= -\sum_{k=1}^{K} \mathbf{T}_{kn} \frac{1}{\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k} \frac{\partial}{\partial \boldsymbol{\Theta}_j} \text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k.
\end{aligned}$$

In order to proceed, we now need to compute the derivative of $\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k$, which depends on the values of j and k, because that dictates whether the derivative of the softmax numerator is nonzero. We'll consider these two cases separately.

If $j \neq k$, then

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\Theta}_j} \text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k &= \frac{\partial}{\partial \boldsymbol{\Theta}_j} \frac{\exp\left(\boldsymbol{\Theta}_k \mathbf{x}^{(n)}\right)}{\sum_{c=1}^{K} \exp\left(\boldsymbol{\Theta}_c \mathbf{x}^{(n)}\right)} \\
&= -\frac{\exp\left(\boldsymbol{\Theta}_k \mathbf{x}^{(n)}\right)}{\left(\sum_{c=1}^{K} \exp\left(\boldsymbol{\Theta}_c \mathbf{x}^{(n)}\right)\right)^2} \frac{\partial}{\partial \boldsymbol{\Theta}_j} \left(\sum_{c=1}^{K} \exp\left(\boldsymbol{\Theta}_c \mathbf{x}^{(n)}\right) \right) \\
&= -\frac{\exp\left(\boldsymbol{\Theta}_k \mathbf{x}^{(n)}\right) \exp\left(\boldsymbol{\Theta}_j \mathbf{x}^{(n)}\right)}{\left(\sum_{c=1}^{K} \exp\left(\boldsymbol{\Theta}_c \mathbf{x}^{(n)}\right)\right)^2} \frac{\partial}{\partial \boldsymbol{\Theta}_j} \left(\boldsymbol{\Theta}_j \mathbf{x}^{(n)} \right) \\
&= -\left(\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k\right) \left(\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_j\right) \frac{\partial}{\partial \boldsymbol{\Theta}_j} \left(\boldsymbol{\Theta}_j \mathbf{x}^{(n)} \right)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial J^{(n)}(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}_j} &= -\sum_{k=1}^{K} \mathbf{T}_{kn} \frac{\left(-\left(\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k\right) \left(\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_j\right) \frac{\partial}{\partial \boldsymbol{\Theta}_j} \left(\boldsymbol{\Theta}_j \mathbf{x}^{(n)} \right) \right)}{\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k} \\
&= \sum_{k=1}^{K} \mathbf{T}_{kn} \left(\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_j\right) \frac{\partial}{\partial \boldsymbol{\Theta}_j} \left(\boldsymbol{\Theta}_j \mathbf{x}^{(n)} \right)
\end{aligned}$$

Otherwise, if $j = k$, then

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\Theta}_k}\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k &= \frac{\partial}{\partial \boldsymbol{\Theta}_k}\frac{\exp\left(\boldsymbol{\Theta}_k\mathbf{x}^{(n)}\right)}{\sum_{c=1}^{K}\exp\left(\boldsymbol{\Theta}_c\mathbf{x}^{(n)}\right)} \\
&= \frac{\exp\left(\boldsymbol{\Theta}_k\mathbf{x}^{(n)}\right)}{\sum_{c=1}^{K}\exp\left(\boldsymbol{\Theta}_c\mathbf{x}^{(n)}\right)}\frac{\partial}{\partial \boldsymbol{\Theta}_j}\left(\boldsymbol{\Theta}_j\mathbf{x}^{(n)}\right) \\
&\quad - \frac{\exp\left(\boldsymbol{\Theta}_k\mathbf{x}^{(n)}\right)\exp\left(\boldsymbol{\Theta}_k\mathbf{x}^{(n)}\right)}{\left(\sum_{c=1}^{K}\exp\left(\boldsymbol{\Theta}_c\mathbf{x}^{(n)}\right)\right)^2}\frac{\partial}{\partial \boldsymbol{\Theta}_j}\left(\boldsymbol{\Theta}_j\mathbf{x}^{(n)}\right) \\
&= \left(\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k - \text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k^2\right)\frac{\partial}{\partial \boldsymbol{\Theta}_j}\left(\boldsymbol{\Theta}_j\mathbf{x}^{(n)}\right) \\
&= \left(\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k\right)\left(1 - \text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k\right)\frac{\partial}{\partial \boldsymbol{\Theta}_j}\left(\boldsymbol{\Theta}_j\mathbf{x}^{(n)}\right)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial J^{(n)}(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}_j} &= -\sum_{k=1}^{K}\mathbf{T}_{kn}\frac{\left(\left(\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k\right)\left(1 - \text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k\right)\frac{\partial}{\partial \boldsymbol{\Theta}_j}\left(\boldsymbol{\Theta}_j\mathbf{x}^{(n)}\right)\right)}{\text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k} \\
&= -\sum_{k=1}^{K}\mathbf{T}_{kn}\left(1 - \text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_j\right)\frac{\partial}{\partial \boldsymbol{\Theta}_j}\left(\boldsymbol{\Theta}_j\mathbf{x}^{(n)}\right)
\end{aligned}$$

We can unify these cases using an indicator variable, $\mathbb{I}[k = j]$, which equals 1 if $k = j$ and 0 otherwise:

$$\frac{\partial J^{(n)}(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}_j} = -\sum_{k=1}^{K}\mathbf{T}_{kn}\left(\mathbb{I}[k = j] - \text{softmax}(\boldsymbol{\Theta}\mathbf{x}^{(n)})_k\right)\frac{\partial}{\partial \boldsymbol{\Theta}_j}\left(\boldsymbol{\Theta}_j\mathbf{x}^{(n)}\right).$$

1.2 Multiclass Classification with Neural Networks

At this point, some of you might be wondering why we've left $\frac{\partial}{\partial \boldsymbol{\Theta}_j}\left(\boldsymbol{\Theta}_j\mathbf{x}^{(n)}\right)$ unsimplified in the entirety of the derivation above (as we all know by now, $\frac{\partial}{\partial \boldsymbol{\Theta}_j}\left(\boldsymbol{\Theta}_j\mathbf{x}^{(n)}\right)$ is just $(\mathbf{x}^{(n)})^T$). While we could have simplified this expression further, leaving it in this form makes the transition to multiclass neural networks almost trivial.

At a high level, all we need to do in order to go from multinomial logistic regression to multiclass classification with neural networks is replace the linear model $\boldsymbol{\Theta}_j\mathbf{x}^{(n)}$ with the output of a (generally but not necessarily) nonlinear neural network $h_{\Theta^{(1)},\ldots,\Theta^{(L)}}(\mathbf{x}^{(n)})$, where $\Theta^{(1)}, \ldots, \Theta^{(L)}$ are the weight matrices in each layer of the neural network.

More formally, if our labels come from one of K classes, then the output layer of our neural network will consist of K nodes, one for each class. We apply a softmax to these outputs

and once again minimize the cross-entropy loss between this and the one-hot encoding of the true label:

$$J^{(n)}(\mathbf{\Theta}) = -\sum_{k=1}^{K} \mathbf{T}_{kn} \log \ \text{softmax}(h_{\Theta^{(1)},\ldots,\Theta^{(L)}}(\mathbf{x}^{(n)}))_k.$$

Following the derivation above, we can conclude that the gradient of this cross-entropy loss w.r.t. $\Theta^{(l)}$ is just

$$\frac{\partial J^{(n)}(\mathbf{\Theta})}{\partial \mathbf{\Theta}^{(l)}} = -\sum_{k=1}^{K} \mathbf{T}_{kn} \left(\mathbb{I}[k=j] - \text{softmax}(h_{\Theta^{(1)},\ldots,\Theta^{(L)}}(\mathbf{x}^{(n)}))_k \right) \frac{\partial}{\partial \mathbf{\Theta}^{(l)}} \left(h_{\Theta^{(1)},\ldots,\Theta^{(L)}}(\mathbf{x}^{(n)}) \right).$$

The two new quantities in this equation, $h_{\Theta^{(1)},\ldots,\Theta^{(L)}}(\mathbf{x}^{(n)})$ and $\frac{\partial}{\partial \mathbf{\Theta}^{(l)}} \left(h_{\Theta^{(1)},\ldots,\Theta^{(L)}}(\mathbf{x}^{(n)}) \right)$, can be computed using forward propagation and backpropagation respectively.