Task 1:

1. $$g^{(c)}(\mathbf{w}_0, ..., \mathbf{w}_{c-1}) = \frac{1}{P}\sum_{i=1}^{P} \log\left(1 + \sum_{j=0, j\neq y_i}^{c-1} e^{\mathbf{x}_i^T(\mathbf{w}_j - \mathbf{w}_{y_i})}\right)$$ Multi-Class Softmax (Ch7 Slide 45)

For the case that $c = 2$, $y_i \in \{0, 1\}$

2. $$y_i = 0 \implies \sum_{j=0, j\neq y_i}^{c-1} e^{\mathbf{x}_i^T(\mathbf{w}_j - \mathbf{w}_{y_i})} = e^{\mathbf{x}_i^T(\mathbf{w}_1 - \mathbf{w}_0)}$$

and

3. $$y_i = 1 \implies \sum_{j=0, j\neq y_i}^{c-1} e^{\mathbf{x}_i^T(\mathbf{w}_j - \mathbf{w}_{y_i})} = e^{\mathbf{x}_i^T(\mathbf{w}_0 - \mathbf{w}_1)}$$

Let $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_0$
If we remap the category index 0 to -1 so that $y_i \in \{-1, 1\}$
Then we can define a compact univariate 2-class Softmax function
that is equivalent to the multivariate case:

4. $$h(\mathbf{w}) = \frac{1}{P}\sum_{i=1}^{P} \log\left(1 + e^{-y_i \mathbf{x}_i^T \mathbf{w}}\right)$$ Two-Class Softmax (eq 6.25)

$\because y_i = -1 \implies e^{-y_i \mathbf{x}_i^T \mathbf{w}} = e^{\mathbf{x}_i^T \mathbf{w}} = e^{\mathbf{x}_i^T(\mathbf{w}_1 - \mathbf{w}_0)} = (2.)$
and $y_i = 1 \implies e^{-y_i \mathbf{x}_i^T \mathbf{w}} = e^{-\mathbf{x}_i^T \mathbf{w}} = e^{\mathbf{x}_i^T(\mathbf{w}_0 - \mathbf{w}_1)} = (3.)$

Thus Multi-Class Softmax can reduce to Two-Class Softmax when $c = 2$.

Task 2:

1. $$g(\mathbf{w}_0, \ldots, \mathbf{w}_{c-1}) = \frac{1}{P}\sum_{i=1}^{P}\left(\log\left(\sum_{j=0}^{c-1} e^{\mathbf{x}_i^T \mathbf{w}_j}\right) - \mathbf{x}_i^T \mathbf{w}_{y_i}\right)$$ Multi-class Softmax (eq 7.23)

2. $$\log\left(\sum_{j=0}^{c-1} e^{\mathbf{x}_i^T \mathbf{w}_j}\right) - \mathbf{x}_i^T \mathbf{w}_{y_i} = \log\left(\sum_{j=0}^{c-1} e^{\mathbf{x}_i^T \mathbf{w}_j}\right) - \log\left(e^{\mathbf{x}_i^T \mathbf{w}_{y_i}}\right)$$ $\log(e^a) = a$

3. $$\log\left(\sum_{j=0}^{c-1} e^{\mathbf{x}_i^T \mathbf{w}_j}\right) - \log\left(e^{\mathbf{x}_i^T \mathbf{w}_{y_i}}\right) = \log\left(\frac{\sum_{j=0}^{c-1} e^{\mathbf{x}_i^T \mathbf{w}_j}}{e^{\mathbf{x}_i^T \mathbf{w}_{y_i}}}\right)$$ $\log(a) - \log(b) = \log(\frac{a}{b})$

4. $$\log\left(\frac{\sum_{j=0}^{c-1} e^{\mathbf{x}_i^T \mathbf{w}_j}}{e^{\mathbf{x}_i^T \mathbf{w}_{y_i}}}\right) = -\log\left(\frac{e^{\mathbf{x}_i^T \mathbf{w}_{y_i}}}{\sum_{j=0}^{c-1} e^{\mathbf{x}_i^T \mathbf{w}_j}}\right)$$ $\log(\frac{a}{b}) = -\log(\frac{b}{a})$

5. $$g(w_0, \ldots, w_{c-1}) = -\frac{1}{P}\sum_{i=1}^{P}\log\left(\frac{e^{\mathbf{x}_i^T \mathbf{w}_{y_i}}}{\sum_{j=0}^{c-1} e^{\mathbf{x}_i^T \mathbf{w}_j}}\right)$$ $\therefore$

In the case $c = 2$, $y_i \in \{0, 1\}$:

6. $$y_i = 1 \implies \frac{e^{\mathbf{x}_i^T \mathbf{w}_{y_i}}}{\sum_{j=0}^{c-1} e^{\mathbf{x}_i^T \mathbf{w}_j}} = \frac{e^{\mathbf{x}_i^T \mathbf{w}_1}}{e^{\mathbf{x}_i^T \mathbf{w}_1} + e^{\mathbf{x}_i^T \mathbf{w}_0}} = \frac{1}{1 + e^{\mathbf{x}_i^T(\mathbf{w}_0 - \mathbf{w}_1)}}$$
$$= \frac{1}{1 + e^{-(\mathbf{x}_i^T(\mathbf{w}_1 - \mathbf{w}_0))}} = \sigma(\mathbf{x}_i^T(\mathbf{w}_1 - \mathbf{w}_0))$$

and likewise:

7. $$y_i = 0 \implies \frac{e^{\mathbf{x}_i^T \mathbf{w}_{y_i}}}{\sum_{j=0}^{c-1} e^{\mathbf{x}_i^T \mathbf{w}_j}} = \sigma(\mathbf{x}_i^T(\mathbf{w}_0 - \mathbf{w}_1))$$
$$= \sigma(-\mathbf{x}_i^T(\mathbf{w}_1 - \mathbf{w}_0)) = 1 - \sigma(\mathbf{x}_i^T(\mathbf{w}_1 - \mathbf{w}_0))$$ $\sigma(-z) = 1 - \sigma(z)$

Let $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_0$

With a clever use of multiplication by 0, we can define a univariate function that is equivalent to the multivariate softmax:

8. $$h(\mathbf{w}) = -\frac{1}{P}\sum_{i=1}^{P} y_i \log\left(\sigma(\mathbf{x}_i^T \mathbf{w})\right) + (1 - y_i)\log\left(1 - \sigma(\mathbf{x}_i^T \mathbf{w})\right)$$ Binary Cross-Entropy (eq 6.12)

$\because y_i = 1 \implies$ the expression in the sum will yield $1 * \log(\sigma(\mathbf{x}_i^T \mathbf{w}))$

and $y_i = 0 \implies$ it will yield $(1 - 0) * \log(1 - \sigma(\mathbf{x}_i^T \mathbf{w}))$

9. $$g(\mathbf{w}_0, \mathbf{w}_1) = h(\mathbf{w})$$ $\therefore$

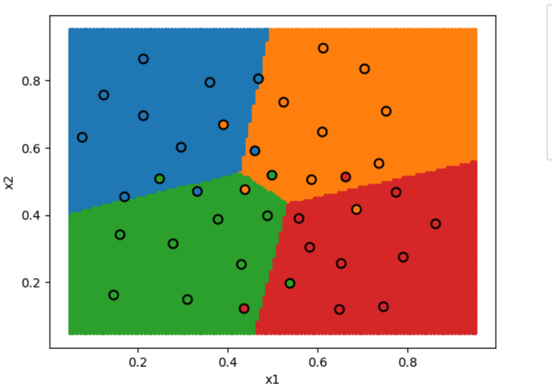
Thus Multi-Class Softmax can reduce to Binary Cross-Entropy when $c = 2$.

Task 3:

Final accuracy: 75% (30/40)

My model uses jax grad() to implement first order local optimization (unnormalized gradient descent). The multi-class model uses the basic Multi-class Softmax for loss with the option of using regularization and normalization. Learning rate is constant equal to 1 because the gradient descent is unnormalized, I let the gradient control the step length. I chose 1000 max iterations to test the different hyperparameter configurations.

Using jnp.ones initialization, no normalization, and no regularization (of weights), the model reached a final loss of 0.48 but it wasn't even done converging at that point, although it did reach the desired accuracy of 75%.



Using jax.random.normal weight initialization, no normalization and no regularization, the model still took a very long time to converge, after 1000 iterations it wasn't yet done, and reached a final loss of 0.48, with final accuracy of 75%. Seemingly the same as before.

Using jax.random.normal weight initialization, no normalization, and with regularization, $\lambda = 0.0001$ the model still took all 1000 iterations, yet achieved worse accuracy, 72%. $\lambda = 0.00001$, restored the 75% accuracy but still no improvement to training time. Losses were slightly higher due to the regularization term.

Using jax.random.normal initialization, with normalization after each gradient update, but no regularization, the model actually converged in 205 iterations, a remarkable improvement, the accuracy was 75%, but interestingly the final loss was higher at 1.18.

Finally, using random initialization, normalization and regularization with $\lambda = 0.001$, the model converged even faster at 177 iterations, with 1.18 loss. The accuracy however never was able to improve beyond 75% for any method I tried.