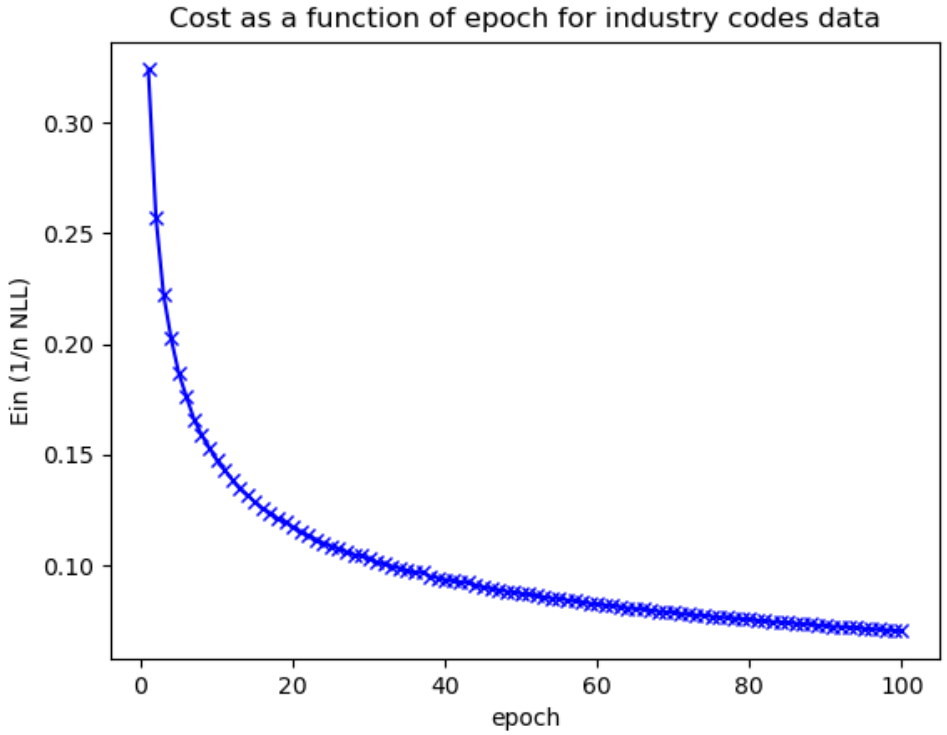
Machine Learning Hand in 1

Group 41: Tan Wei Yuan, Ang Yu Juan, Alton ZhiXian Cheah

PART I: Logistic Regression

Code

Our logistic regression function resulted in a 0.979 insample error and a test-score of 0.963. After some testing, we decided to use learning rate 0.1, batch-size 20 with 100 epochs as it gave us the most accurate results. There are no anomalies from the plot as the cost function decreases as how we expected it to.



Our cost_grad and fit functions follow the lecture slides.

```
def cost_grad(self, X, y, w):
    cost = 0
    grad = np.zeros(w.shape)
    ### YOUR CODE HERE
    grad = - 1/len(y) * np.matmul(logistic(- np.dot(X, w)* y) *y, X)
    cost = 1/len(y) * np.sum(np.log(1 + np.exp(- np.dot(X, w) * y)))
    ### END CODE
    assert grad.shape == w.shape
    return cost, grad
```

```python
def fit(self, X, y, w=None, lr=0.1, batch_size=16, epochs=10):
    if w is None: w = np.zeros(X.shape[1])
    history = []
    ### YOUR CODE HERE
    for i in range(epochs):
        perm = np.random.permutation(len(y))
        X = X[perm]
        y = y[perm]
        for j in range(0, len(y), batch_size):
            X_b = X[j:j+batch_size]
            y_b = y[j:j+batch_size]
            #Getting gradient
            cost, grad = self.cost_grad(X_b, y_b, w)
            w = w - lr*grad
        #Getting cost
        cost, grad = self.cost_grad(X, y, w)
        history.append(cost)
    ### END CODE
    self.w = w
    self.history = history
```

Theory

1.Time complexity

We can evaluate the complexity by looking at the fit function

The outer loop runs epoch times, the inner loop runs n/batch-size times, each time running the cost_grad function.

Complexity for the cost_grad function:

Both cost and grad takes around O(batch-size * d) time due to the dot product

For both cost and grad, Overall time complexity O(epoch * n * d)

2.Sanity Check

Randomly permutating the data will result in worse performing classifier. A random permutation disrupts the spatial relationships between pixels, making it harder for the model to extract meaningful features and patterns, which could hinder its performance.

3.Linearly Separable Data

When the data is linearly separable, the weights are unbounded and can approach infinity in magnitude as there can be a decision boundary that separates all data points correctly. The weights become larger and the decision boundary sharper.

PART II: Softmax

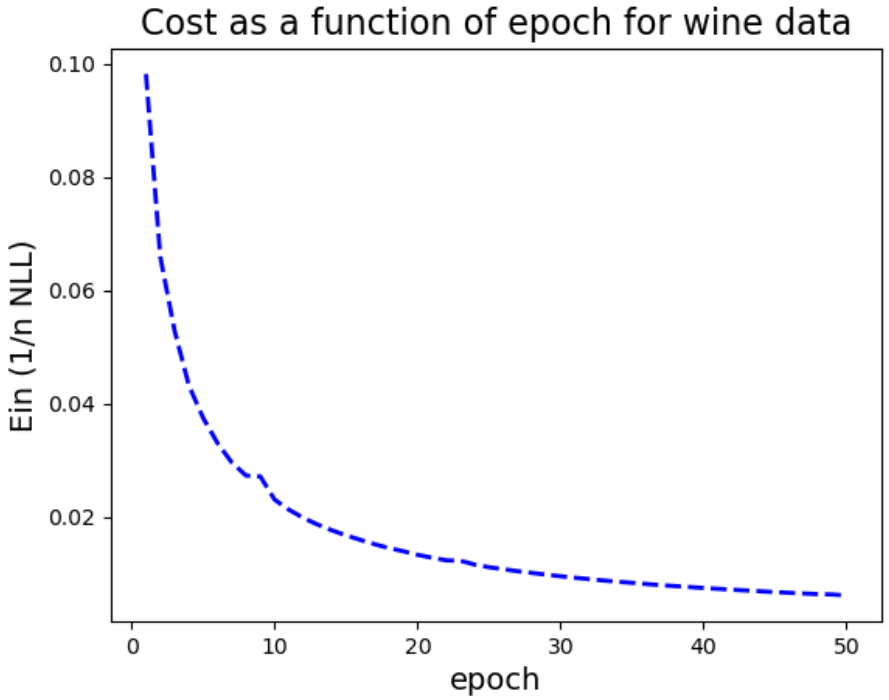
Code

Our softmax function uses similar fit function as logistic regression, with the cost and grad changed to follow the lecture notes.

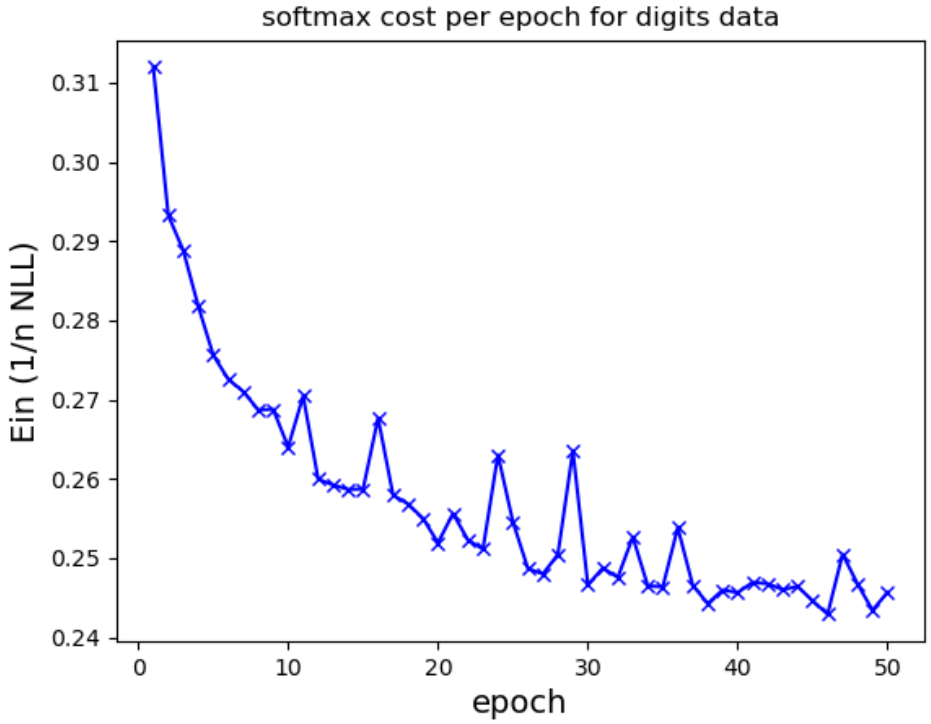
```python
def cost_grad(self, X, y, W):

    cost = np.nan
    grad = np.zeros(W.shape)*n
    Yk = one_in_k_encoding(y, self.num_classes) # may help - otherwise you may remove it
    ### YOUR CODE HERE
    cost = -1/len(y) * np.sum(Yk * np.log(softmax(np.dot(X,W)))) #ndk
    grad = -1/len(y) * np.dot(X.T,(Yk-softmax(np.dot(X,W))))  #ndk
    ### END CODE
    return cost, grad
```

(parameter) self: Self@SoftmaxClassifier

```python
def fit(self, X, Y, W=None, lr=0.01, epochs=10, batch_size=16):

    if W is None: W = np.zeros((X.shape[1], self.num_classes))
    history = []
    ### YOUR CODE HERE
    for i in range(epochs):
        perm = np.random.permutation(len(Y))
        X = X[perm]
        Y = Y[perm]

        for j in range(0, len(Y), batch_size):
            X_b = X[j:j+batch_size]
            Y_b = Y[j:j+batch_size]
            cost, grad = self.cost_grad(X_b, Y_b, W)

            W = W - lr*grad
        cost, grad = self.cost_grad(X, Y, W)
        history.append(cost)
    ### END CODE
    self.W = W
    self.history = history
```



Our wine data gives an insample accuracy of 1 and an outsample accuracy of 0.977. We tried some tuning but it did not affect the accuracies much. For this graph we used learning rate 0.5, batch size 10, with 50 epochs



We settled on learning rate 0.1, batch size 20, epochs 50 for our digits data. This gave us an insample accuracy of 0.931 and a test accuracy of 0.921. The graph shows some small spikes in error which might be due to 0.1 being still too high learning rate, but the overall error trends downwards.

Theory

The main difference between the time complexity of our softmax and that of our logistic regression lies in the weight matrix W being (n,k). During the cost_grad, the dot product of X and W gives an O(epocch * d * k) time complexity. Using a same fit function, the overall time complexity becomes O(epoch * n * d * k)