Machine Learning Handin 1

Erik Holm Steenberg (201804655), Hans Brüner Dein (201706079),
Camilla Theresia Grøn Sørensen (201807007)
Instruktor: Mads Bech Toftrop

October 2022

1 Introduction

This is the report for the 1st handin in the computer science course Machine Learning. The handin had two parts, one part about logistic regression and one part about softmax. Each section is split into a code part and a theory part. Accuracy results have been rounded down for sanity reasons.

2 PART I: Logistic Regression

2.1 Code

We have used the following code for our **cost_grad** and **fit**:

```
def cost_grad(self, X, y, w):
    cost = 0
    grad = np.zeros(w.shape)
    for i in range (X.shape[0]):
        cost += np.log(1+np.exp(-y[i]*np.dot(X[i,:],w)))
        grad += -y[i]*X[i,:]*logistic(-y[i]*np.dot(X[i,:],w))
    cost = cost/X.shape[0]
    grad = grad/X.shape[0]
    assert grad.shape == w.shape
    return cost, grad

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 = []
    for i in range(epochs):
        Xy=np.random.permutation(Xy)
        for j in range(0,len(Xy),batch_size):
            XyB=Xy[j:j+batch_size,:]
            XB=XyB[:,:-1]
            yB=XyB[:,-1]
            grad=self.cost_grad(XB,yB,w)[1]
            w=w-lr*grad
        history.append(self.cost_grad(X,y,w)[0])
    self.w = w
    self.history = history
```

Summary and Results

We got the following plot

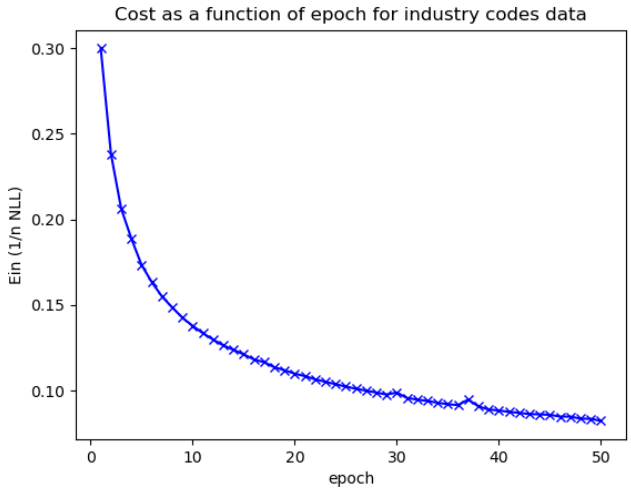

Figure 1: The plot shows E_{in} as a function of epochs. Generally E_{in} becomes better as a function of epochs, but for two epochs it becomes worse. This is probably because the gradient descent overshoots for those two epochs.

The accuracy that we achieved for the Sample Score was 0.97 and for Test Score it was 0.96. Which is very nice.

2.2 Theory

1

The time order of the mini_batch is $O(\frac{epochs \cdot n^2 d}{batch\_size})$. The time order for the cost and the gradient for the cost_grad code is $O(nd)$ and $O(nd)$ respectively.

2

Logistic Regression takes a list of features for each data point. For the case with pictures, each pixel is a feature. By permuting each picture the same way, we permute the list of features. This influences the locality, but Logistic Regression doesn't take locality into account in the first place. So the classifier becomes neither better or worse by permuting the pixels of each picture.

3

If the data is linearly separable then there exists an explicit solution with an error of zero, e.g. $Xw - y = 0$. By running gradient descent as long as we want, the weights that minimize the negative log likelihood becomes

$$w = X^{-1}y \tag{1}$$

Where X is the data matrix and y the result.

3 PART II: Softmax

3.1 Code

Our implementation of the **cost_grad** and **fit** are as follows:

```
def cost_grad(self, X, y, W):
    cost = 0
    grad = np.zeros(W.shape)
    Yk = one_in_k_encoding(y, self.num_classes)
    n = X.shape[0]
    XW = softmax(X@W)
    cost=-1/n*np.sum(np.log(np.sum(Yk*XW,axis=1)))
    grad=-1/n*X.T@(Yk-XW)
    return cost, grad

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 = []
    Xy=np.c_[X, Y]
    for i in range(epochs):
        Xy=np.random.permutation(Xy)
        for j in range(0,len(Xy),batch_size):
            XyB=Xy[j:j+batch_size,:]
            XB=XyB[:,:-1]
            yB=XyB[:,-1]
            W=W-lr*self.cost_grad(XB,yB,W)[1]
        history.append(self.cost_grad(X,Y,W)[0])
    self.W = W
    self.history = history
```

Summary and Results

Applying the code we get the following plots

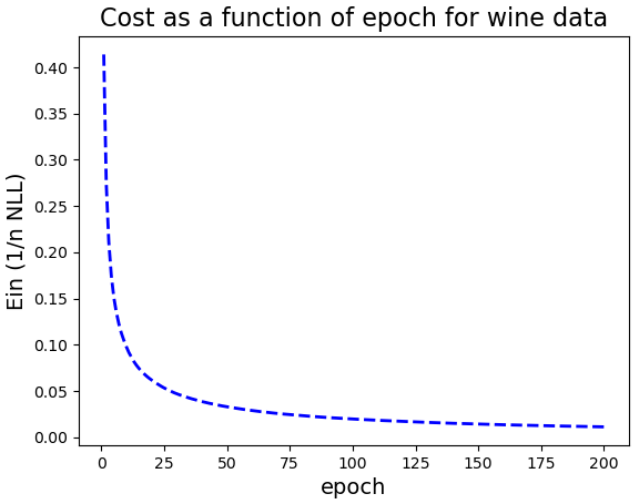

Figure 2: E_{in} as a function of epochs for the classification of wine. It can be seen that E_{in} is minimized quickly.

The accuracy that we achieved for the Sample Score was 1.0 and for Test Score was 0.97

Show-digits

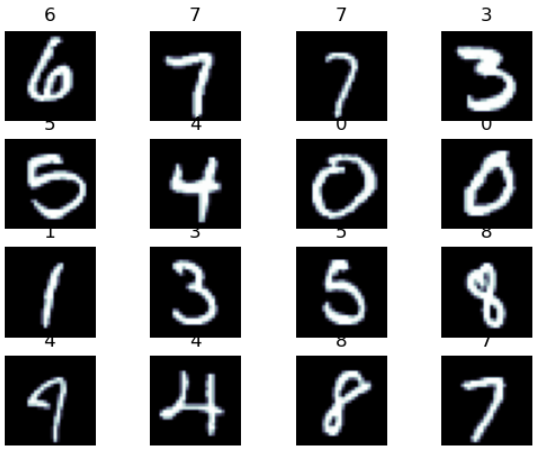

Figure 3: This is an example of training data.

Digits

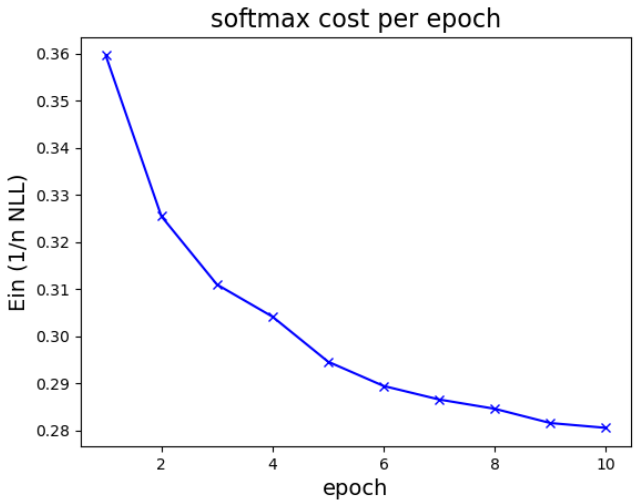

Figure 4: E_{in} is given as a function of epochs. We only do 10 epochs, and our E_{in} does not become better than 0.28.

The accuracy that we achieved for the Sample Score was 0.92 and for Test Score it was 0.92 as well.

Visualize

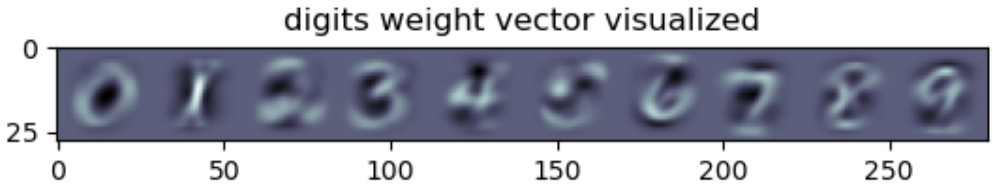

Figure 5: This figure is the weights for each number. In words, this is what code views as essential for each number. It is was the code views as being necessary if what you write has to qualify as a number.

3.2 Theory

The time order of the cost/grad function is limited by the two products of $d \times n \cdot n \times k$ and $n \times d \cdot d \times k$ matrices, which can both be computed in $O(nkd)$ time. Since we use the same code for the mini-batch gradient decent the total time is $O(\frac{epochs \cdot n^2 kd}{batch\_size})$