Bryan Bezerra - bezerra
Selimhan Erhan - se454
Sihua Zhou - sz583

Project 3: You Always Cut the Red Wire

Group Members: Selimhan Erhan (se454), Bryan Bezerra , Sihua Zhou (sz583)
P.S: Selimhan Erhan discussed with TA Patrick Meng during 1 on 1 appointment on December 13th. All of us were dedicated to work on all the parts via Github.

Background:

- It is another day on the deep space vessel Archaeopteryx, and someone has accidentally activated the self-destruct mechanism. It's up to you to save the ship. Fortunately, you have the manual.
- The manual contains pages and pages of complex wiring diagrams, each labeled as 'safe' or 'dangerous', and for the 'dangerous' ones, which wire needs to be cut. But there is no general rule listed in the manual - you'll need to try to learn what makes a wiring safe or dangerous, and what needs cutting, based on the examples in the manual.

First Task

Input Space

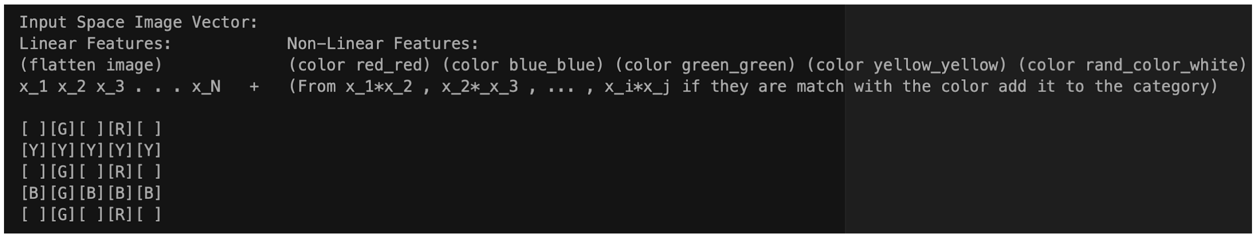
We decided to use one-hot encoding for our input space. Each pixel of the bomb diagram image is represented by a vector of length four, and each element of the vector represents one color that a wire can be. The possible colors are as follows:

- Green – [1, 0, 0, 0]
- Yellow – [0, 1, 0, 0]
- Blue – [0, 0, 1, 0]
- Red – [0, 0, 0, 1]
- No wire – [0, 0, 0, 0]

These pixel vectors are then flattened into one large vector containing all the pixels in the image, and one extra element set to 1 is added to the front of the vector. For a 20x20 bomb diagram, this results in a vector of length 1601.

Adding Non-Linear Features on Top of Linear Feature (phi(x)):

Description of Non-Linear Features:

- Since this part is identifying whether a graph is dangerous or not, then it is worth to check how two colors are neared to each other and count the number of colors (i.e. dot product of cell i and cell j = 1 and check which color it belong to). Worth notice white color dot with any other color would be equal to 0.
- 

```
Input Space Image Vector:
Linear Features:             Non-Linear Features:
(flatten image)              (color red_red) (color blue_blue) (color green_green) (color yellow_yellow) (color rand_color_white)
x_1 x_2 x_3 . . . x_N   +   (From x_1*x_2 , x_2*_x_3 , ... , x_i*x_j if they are match with the color add it to the category)

[ ][G][ ][R][ ]
[Y][Y][Y][Y][Y]
[ ][G][ ][R][ ]
[B][G][B][B][B]
[ ][G][ ][R][ ]
```

Bryan Bezerra - bezerra
Selimhan Erhan - se454
Sihua Zhou - sz583

- In addition, it would result in vector of length 1601 + flat_image_size – 1 = 1601 + 1600 – 1 = 3200

Output Space

The output space consists of two integers, 0 and 1. 0 indicates a safe wiring diagram, while a 1 means it is dangerous.

Model Space

As this is a binary classification problem, we decided to use the non-linear function:

$$f_{\underline{w}}(\underline{x}) = \sigma(\underline{w} \bullet \underline{x}) = 1 / (1 + e^{(\underline{w} \bullet \underline{x})})$$

for our predictions. The vector of weights is given by $\underline{w}$, and the input vector is given by $\underline{x}$. The function always produces values between 0 and 1, making it easy to use it to represent probabilities. To ensure that predictions are binary, values < 0.5 are returned as a 0, and values >= 0.5 are returned as 1.

Measuring Loss

We chose the log loss function to measure the error of a given model:

$$\text{Loss}(f_{\underline{w}}(\underline{x}), y) = -y\ln[f_{\underline{w}}(\underline{x})] - (1-y)\ln[1 - f_{\underline{w}}(\underline{x})]$$

Training and Overfitting

We chose stochastic gradient descent with ridge regularization to train our model. For the 0th step, each weight is randomly set to a value between negative and positive 1 / sqrt(1600) ensuring that starting weights are proportional to the size of the vector and are not too big, risking overfitting. For each successive step, we use stochastic gradient descent to improve our model:

$$\underline{w}(k+1) = \underline{w}(k) - \alpha\,([f_{\underline{w}(k)}(\underline{x}) - y]\underline{x} + 2\lambda\boldsymbol{\Sigma}\underline{w})$$

The $[f_{\underline{w}(k)}(\underline{x}) - y]\underline{x}$ is the derivative of the loss function with respect to w_j. By minimizing this value, we get closer and closer to minimizing the overall loss of the model. The constant, α, is a value that controls how big each step is. If it is too low, the model takes too long to train, but if it is too large, the model may skip over the optimal weights. Finally, $2\lambda\boldsymbol{\Sigma}\underline{w}$ is a penalty added to the loss to prevent weights from becoming too large and therefore overfitting. The magnitude of this ridge regularization is controlled by the constant λ.

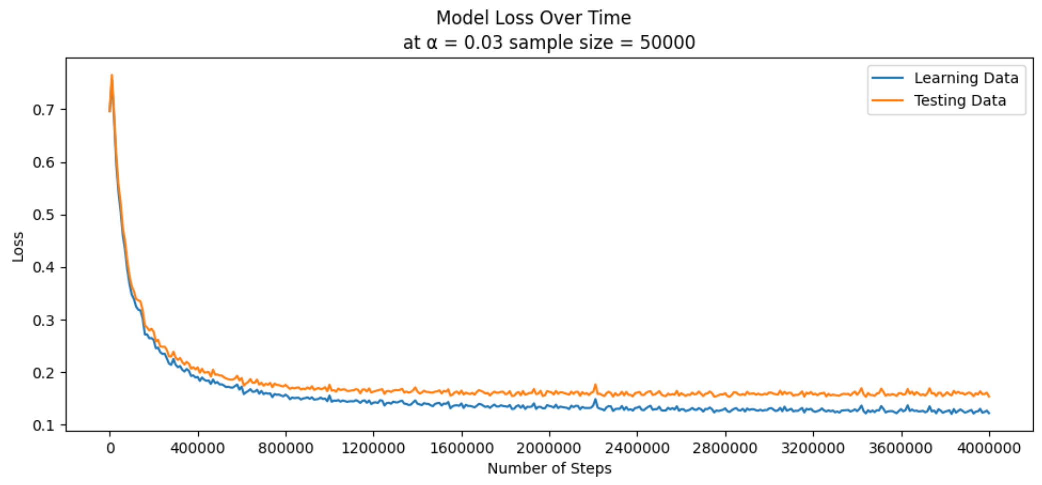
Performance

As expected, the model's performance on data it had not seen before increased with the size of the set of samples fed to it:

# of Samples	**Min Training Data Loss**	**Min Testing Data Loss**
2,000	0.0385	0.5634
2,500	0.0359	0.4860
3,000	0.0288	0.4085
5,000	0.0354	0.2624
50,000	0.1240	0.1583

Bryan Bezerra - bezerra
Selimhan Erhan - se454
Sihua Zhou - sz583

At these small sample sizes, the training data loss is dramatically lower than the training data loss, indicating that a great deal of overfitting is happening even with ridge regularization. The situation improves with more samples. The testing data loss is brought much closer to the training data loss when the sample size is raised by a factor of 10:



As can be seen in the figure above, with a larger sample size, the learning data loss and the testing data loss stay close in value over time as the model is trained, suggesting that the model is not overfitting the training data. The model's minimum loss values have a difference of only 0.0342, and the model provides much more accurate predictions on the test data it has not yet seen than models trained on fewer samples.

Justification for Non-Linear Feature:

- We consider the idea for non-linear features is reasonable however might not really help with improving the loss. One of the reasons could be too many features added in to one vector and other one could the vector of raw data is good enough for training. Off this case, it is highly likely that the raw data is self is good enough to cover most of the datapoint. Since adding more non-linear feature would result more curves in graph (up and down).

Assessment:

- According to the graph we obtained, I believe we failed with having a good fit and failed to prevent overfit even we tried implment regularization in the program. Reason that we can think of are failed to early terminate and use better algorthimm that fit for this type of training, for example, use very difference of the learning and testing loss.

Second Task

Input Space

We decided to use one-hot encoding for our input space. Each pixel of the bomb diagram image is represented by a vector of length four, and each element of the vector represents one color that a wire can be. The possible colors are as follows:

- Green – [1, 0, 0, 0]
- Yellow – [0, 1, 0, 0]

Bryan Bezerra - bezerra
Selimhan Erhan - se454
Sihua Zhou - sz583

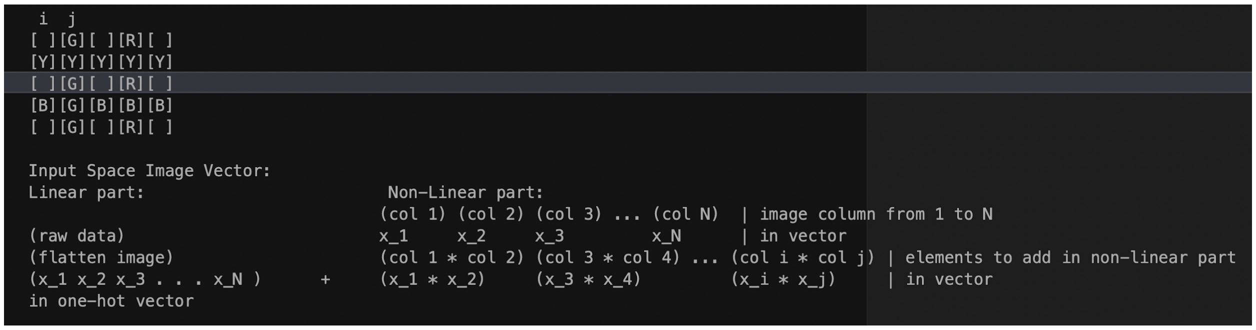
- Blue – [0, 0, 1, 0]
- Red – [0, 0, 0, 1]
- No wire – [0, 0, 0, 0]

These pixel vectors are then flattened into one large vector containing all the pixels in the image, and one extra element set to 1 is added to the front of the vector. For a 20x20 bomb diagram, this results in a vector of length 1601.

Adding Non-Linear Features on Top of Linear Feature (phi(x)):

Description of Non-Linear Features:

- Since this part is asking for which wire to cut it is useful to consider the column of the image and find the relationship between columns which is of vertical wires. (i.e. dot product of every two columns column i and column j which is adjacent to each other)



```
  i  j
[ ][G][ ][R][ ]
[Y][Y][Y][Y][Y]
[ ][G][ ][R][ ]
[B][G][B][B][B]
[ ][G][ ][R][ ]

Input Space Image Vector:
Linear part:                        Non-Linear part:
                                   (col 1) (col 2) (col 3) ... (col N)  | image column from 1 to N
(raw data)                         x_1     x_2     x_3         x_N      | in vector
(flatten image)                    (col 1 * col 2) (col 3 * col 4) ... (col i * col j) | elements to add in non-linear part
(x_1 x_2 x_3 . . . x_N )      +    (x_1 * x_2)     (x_3 * x_4)         (x_i * x_j)     | in vector
in one-hot vector
```

-
- In addition, it would result in a vector of length 1601 + diagram_size / 2 = 1601 + 10 = 1611

Output Space

The output space is a four-dimensional output vector, where each element represents the softmax probability for the corresponding class, each dimension representing the color of the wire that is needed to be cut. [P(Blue) , P(Red) , P(Green) , P(Yellow)]

Model Space

As this is a non-binary function, we decided to use a Multi-Class Classification (softmax function):

$$\left(p_1, p_2, \dots, p_c\right) = \left(\frac{e^{\underline{w}^1 \underline{x}}}{\sum_{d=1}^{C} e^{\underline{w}^d \underline{x}}}, \frac{e^{\underline{w}^2 \underline{x}}}{\sum_{d=1}^{C} e^{\underline{w}^d \underline{x}}}, \frac{e^{\underline{w}^3 \underline{x}}}{\sum_{d=1}^{C} e^{\underline{w}^d \underline{x}}}, \dots, \frac{e^{\underline{w}^C \underline{x}}}{\sum_{d=1}^{C} e^{\underline{w}^d \underline{x}}}\right)$$

Given an input vector x, we want to produce an output vector of C many probabilities, each entry pc corresponding to the probability that x is in class c. We can generalize the effect of the sigmoid function with the Softmax function. For each class c, let w_c be a weight vector corresponding to that class (again taking the 0th component as 1).

Bryan Bezerra - bezerra
Selimhan Erhan - se454
Sihua Zhou - sz583

Measuring Loss

We chose the categorical cross-entropy loss to measure the error of a given model:

$$Loss(\underline{p}, \underline{q}) = - \sum_{c=1}^{C} q_c \ln p_c$$

Distribution across class labels, $\underline{y} = (q_1, q_2, q_3, ..., q_C)$.

Training and Overfitting

We chose stochastic gradient descent for training a SoftMax regression model in the context of a multiclass classification problem. The function iteratively updates the model weights based on randomly selected data points from the training set. For each iteration, it computes the softmax regression predictions for the selected data point and calculates the categorical cross-entropy loss by comparing the predicted probabilities with the true labels.

$$w_i \leftarrow w_i - \alpha \frac{\partial L}{\partial w_i}$$

Where w_i is the parameter for the ith class, α is the learning rate, $\frac{\partial L}{\partial w_i}$ is the partial derivative of the loss function L with respect to the weight w_i.

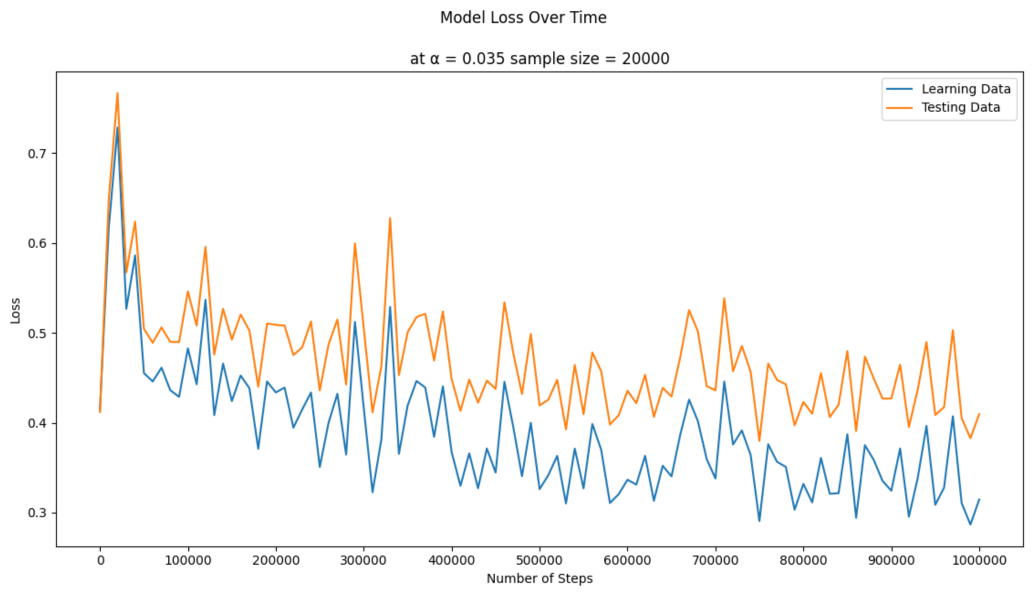
The model's weights are then adjusted using the stochastic gradient descent update rule. Additionally, the function monitors the training progress by recording and printing the training and testing losses every 1000 steps. This process continues for the specified number of training steps, resulting in a trained softmax regression model with updated weights that aim to minimize the categorical cross-entropy loss on the training data. The recorded losses provide insights into the model's performance on both the training and testing datasets throughout the training process.

Performance

As expected, the model's performance on data it had not seen before increased with the size of the set of samples fed to it:

# of Samples	Min Training Data Loss	Min Testing Data Loss
2,000	0.0064	0.3708
2,500	0.0105	0.4058
3,000	0.0186	0.3651
5,000	0.1318	0.3669
50,000	0.3550	0.3891

Bryan Bezerra - bezerra
Selimhan Erhan - se454
Sihua Zhou - sz583



Justification for Non-Linear Feature:

- As mentioned earlier in part 1, adding more non-linear features resulting the graph getting more curve (i.e. goes up and down) for fitting more datapoints. The graph shows this attribute. However, this is not a good sign for testing loss, meaning non-linear feature make the model trying to fit more points and loss the ability to generalize resulting testing loss frequently goes up and down.

The way we prevent overfitting is using L2 regularization, L2 regularization adds a penalty term to the loss function based on the squared magnitudes of the model's weights. This term discourages the weights from becoming too large during training, which prevents overfitting.

$$\lambda\sum i{=}1n \sum_{i=1}^{n} w_i^2$$

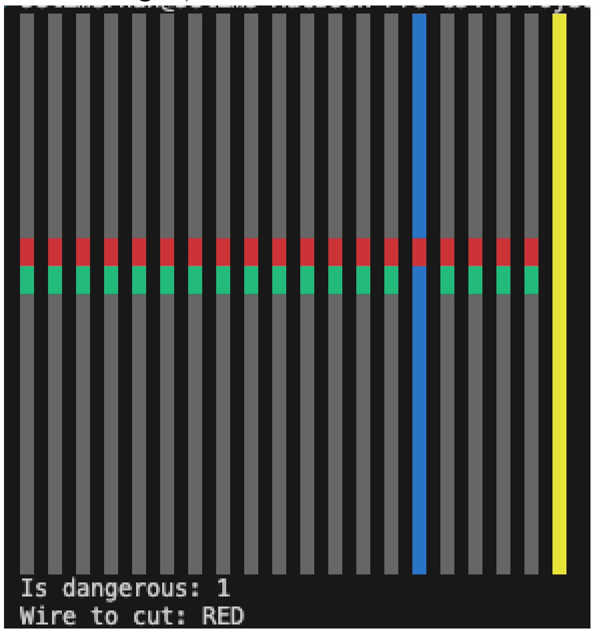
where λ = regularization parameter, n = total number of weights in the model and w_i = represents an individual weight in the model

Assessment:

- Similar to part 1 we used the same way of preventing overfitting, however, in this case from the graph, there is a tendency of converging afterall, dislike the graph in part 1 that are consistently diverge and tend going as straight line. I believe in this case we have made some progress on preventing overfitting from happening.

Bryan Bezerra - bezerra
Selimhan Erhan - se454
Sihua Zhou - sz583

20x20 Image :)



```
Is dangerous: 1
Wire to cut: RED
```