[CS440] Intro to AI Project 3 Report

Arjun Patel (ap2015), Sai Manukonda (skm170)

December 19, 2023

1 The input space and feature selection

We ended up with good results while using the same input space for both of the problems. We spent the most time picking and tuning features and went through several iterations. In the beginning, the input space was a number between 0 and 4 where 0 represented a red cell, 1 represented a blue cell, 2 represented a green cell, 3 represented a yellow cell and 4 represented a white cell. This was a very naive set of features and was quickly abandoned for the one hot encoding of 4-dimensional vectors where each position holds the number of Red, Blue, Green and Yellow pixels in a given observation space in that order. In effect, a vector related to a single pixel will have at max one one of its positions set to 1. Notice that a White pixel will be represented as a zero vector. Right off the bat, saw a minor improvement during our training. However, this improvement in training did no convert to better predictions over testing. This discrepancy was a result of over fitting as the number of features increased from 400 in the first approach to 1600 in the second approach.

Next, we decided to combine the 4-dimensional vector approach with a more refined approach which involves figuring out further information about intersections in the image. Intersections provide spatial information to the model and can be captured by pooling a few adjacent pixels into one vector and passing it into the model. In our model pooling is done over a 2×2 collection of cells where the resulting vector is a sum of all the vectors of each pixel in the 2×2 observation space. A 2×2 observation space is sufficient because each intersection point captured by a 2×2 window will have a ratio of $2 : 1$ between the number of pixels of the wire overlapping the wire underneath and the wire being overlapped (drawing an intersection out makes this apparent). This information can be passed along with a vector for each pixel in the layout so that the model can make a more informed decision. We tested both max pooling and average pooling. In the case of max pooling, we simply assumed that a 2×2 observation space is fully colored a certain color if the count of that color is the highest in the vector obtained by pooling. Essentially, the vector representing the observation space in question will be that of a single colored pixel if that colors occurs the most times in the observation space. In case of average pooling, the vector of a 2×2 observation space will be an average of the sum of vectors in the observation space. Besides simply trying a different approach, this approach normalizes the vector returned, potentially improving the accuracy of the model.

Testing with pooling did not show significant improvement in the model and we continued running into issues with overfitting which makes sense considering that the number of features obtained by combining the 4-dimensional vector method with pooling balloons the number of features from 1600 to 2000. We further pondered over how to capture intersections better and noticed that each intersection follows the following pattern:

1. They are three tiles long
2. If a wire is being overlapped by another wire, then at one point along that wire there will be one spot of a different color. This reduces to a $A-B-A$ pattern along the wire that is being overlapped where A and B are two distinct colors.
3. The number of distinct $A-B-A$ patterns in our data set are 4×3 because we can pick 4 different colors to populate A and then we have 3 leftover to populate B.

This observation made apparent how easy it would be to pass in more accurate information about intersections. Information about all the intersections was computed by looping over all 3 block long sections in the image (vertically and horizontally) and looking for all 12 $A - B - A$ patterns. A 12-dimensional vector was created, where each position held the count of the number of the kind of intersection observed in the image, and was passed into the one hot encoding of the features. Note that this is a non-linear feature because it combines the information of multiple cells (3 cells in this case) into a single value which is then passed as input.

We tested this approach with the 4-dimensional vector approach yielding us 1612 features and saw promising results. Throwing pooling into the mix did not improve the model much. Hence, that approach was abandoned to combat overfitting. Another optimization noticed was that overlaps can be 4 blocks long too in the case where 2 adjacent wires are being over lapped by another wire. Moreover, the information for this kind of overlaps would not be captured by any of thr 3 block long overlap checks because we the pattern $A - B - C - A$. The information for these kinds of overlaps were encoded into a exactly the same way as that described for 3 block long overlaps so we will not go into details here. The 4 block long intersection optimization allowed for a little improvement in our model and we ended up with 1636 features.

At this point, the model for task 1 was getting getting $90 - 95\%$ accuracy in its predictions and we felt satisfied with this model and moved onto task 2 with the same feature set. Note that the model for task 1 was still running into issues with overfitting which was apparent from the validation loss diverging from the training loss and plateauing at a little higher loss value. The model for task 2 did a little bit worse with the this feature set, about $75 - 85\%$ accuracy with overfitting. To combat overfitting and try to make our models better, we decided to reduce the feature set by one hot encoding the a 4-dimensional vector where each element is the count of how many of the corresponding color is present in the image instead of passing in a single 4-dimensional vector for each pixel. This reduced the number of features in our model to 40 features. We ported this feature set into both the tasks and saw amazing results which will be discussed in the data and analysis section of both the tasks.

2 Task 1

2.1 The output space

The model is trying to determine if a given wire layout is dangerous or not. This can be abstracted to a binary space where 0 stands for a not dangerous layout and 1 for a dangerous layout.

2.2 The model space

This is a classification problem and the output space is binary. The Sigmoid function maps this kind of output space the best because this function returns a value between 0 and 1 which can be interpreted as the probability of some event. Hence, we will be using a logistical regression model.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

In effect, once the weights have been determined, the input to the Sigmoid function above, x, will be the dot product of the input vector and the weights determined so far for each element in the input vector. This will return a scalar value which we can simply plug in into the sigmoid function and we will obtain a value between 0 and 1. If the value is less than 0.5, then the image will be classified is not dangerous, otherwise, it will be classified as dangerous.

2.3 The loss function

The function used to measure the loss of a model is the log loss entropy function:

$$Loss(a, y) = \frac{1}{m} \sum_{i=1}^{m} -y * \log a - (1 - y) * \log(1 - a)$$

Where m is the total number of data points considered, y is the expected output of an input and a is the prediction of the model that has been created so far. Hence, a is a result of the sigmoid of the dot product of the weights created by the model so far and the input. In our implementation, for each iteration, the loss of the model is summed up, and at the end of each epoch the average loss is reported to keep track of how the model is doing as training progresses. This is paired with the loss of a validation set so that overfitting can be dealt with.

2.4 The training algorithm

The training algorithm used to train the model over the data is the Stochastic Gradient Descent algorithm. This algorithm was chosen because it is easy to implement and faster than Gradient Descent training algorithm because it does not make us compute the gradient of all the data points as we will have to if gradient descent was used. A pit fall of this algorithm is that updating the weights over a single data point at a time which may make the model good at a few similar data points and worse at the ones not trained on. However, computing weight updates over a lot of iterations will average this overfitting out. The algorithm implemented is as follows. Set the initial weights to zero. For each step, update the weight vector to be the difference between the current weight vector and the product of the learning rate which determines how large the changes will be, the difference between the current prediction of the weight vector given the current data point and the expected value related to the data point and the input vector. Mathematically, this becomes

$$w_{k+1} = w_k - \alpha(a - y)X$$

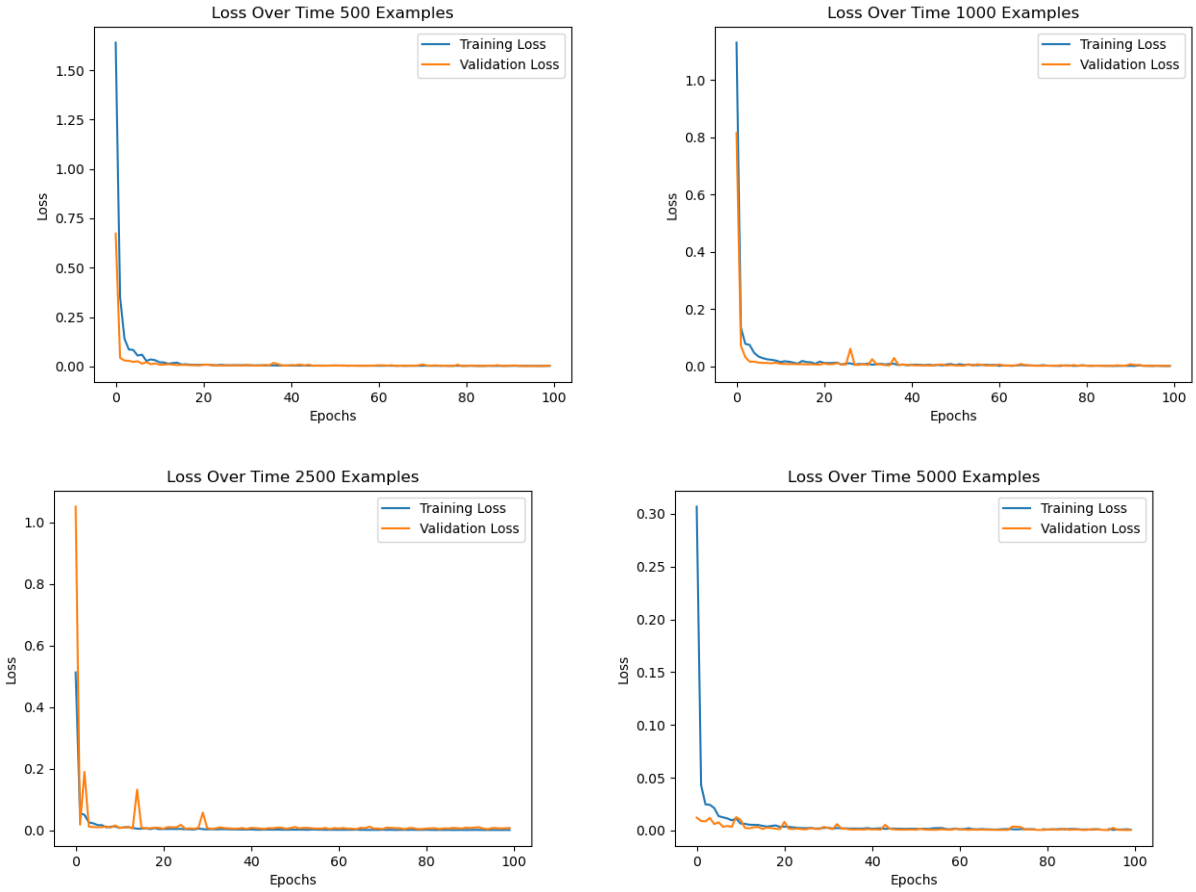
where X is the input vector of the current data point, y is the expected result of the current data point and a is the sigmoid of the dot product of the current weight vector and the input vector of the current data point, also known as the current prediction.

2.5 Dealing with overfitting

A hold out set for early stopping and data augmentation were used to combat over fitting. Data augmentation was implemented by rotating each generated data point four times and placing these 'new' images back into the training data as additional data for the model to train on. This essentially added more data for our model to train on allowing it to generalize better. The essentials for early stopping were implemented but were not utilized. This includes generating a validation set and reporting the loss over the validation set at the end of each epoch. Since the validation loss curve did not show signs of over fitting (a marked increase in the loss of the validation set after one point in the training), early stopping was not included. Note that a cheap and easy way to implemented early stopping would be to stop at the moment the loss of the validation in the current epoch is greater than the previous epoch. All in all, overfitting did not seem to be an issue for task 1.

2.6 Data and analysis

All the data was collected over a learning rate of .01 over 100 training epochs. The total data sample was broken down 90% for training and 10% for validation. Moreover, a separate data set of 100 samples was generated to test the accuracy of the models. The models do not demonstrate overfitting because the validation loss does not diverge from the training loss. Early stopping would be helpful in the case of overfitting and details of how it can be added to our current implementation have been included in the 'Dealing with overfitting' section of task1. The testing of the models showed promising results with the models correctly categorizing an image as dangerous or not 100% of the time when trained over 500 examples, 100% of the time when trained over 1000 examples, 100% of the time when trained over 2500 examples and 100% of the time when trained over 5000 examples. Data can be generated by running python3 Task1.py.



The models seem to capture the problem pretty well according to the data because validation loss stays close to the training loss and model predictions on a given input are highly accurate.

3 Task 2

3.1 The output space

We are trying to figure out which of the four wires to cut. Hence, the output space should be an a abstraction of the four wires. Additionally, only one of the wires can be laid second to last which means that only one of the four wires can be the right answers. Hence, this is an exclusive multi class classification problem which can be represented by assigning the numbers 0 to 3 to each wires. However, to make our lives easier, we decided to represent which wire to cut with a vector wherein all its position sum up to 1 and only 1 of its positions is set to 1 to represent that the wire corresponding to the position of 1 in the vector is to be cut. This makes implementation easier because when calculating the loss of a particular set of weights (**set** of weights because this is a classification problem), the loss for each category (a vector) can be easily calculated by multiplying the log of the prediction in the i^{th} position with i^{th} of the correct classification (0 means do not cut and 1 means cut). In sum, our output space is a number 0 to 3 which corresponds to the wire that is to be cut. However, internally, the classification is handled as a vector.

3.2 The model space

The problem at hand is an exclusive multi class classification problem. The core of solving this problem will be determining how confident the model is that certian input be classified as a certain out put. This is best modeled by Softmax Regression wherein a weights are assigned to each category which when dotted with a particular input output the probability of the input being classified as a certain thing in

the output space. Notice that since we are dealing with probabilities, the sum of all the probabilities of each classification must be 1. The probabilities are determined by the following function:

$$A(a_1, a_2, \ldots, a_n) = (\frac{e^{w_1 \cdot x}}{\sum_{i=1}^{c} e^{w_d \cdot x}}, \frac{e^{w_2 \cdot x}}{\sum_{i=1}^{c} e^{w_d \cdot x}}, \ldots, \frac{e^{w_n \cdot x}}{\sum_{i=1}^{c} e^{w_d \cdot x}})$$

where A is a vector of the probabilities of each class being the answer, w_j being the weight corresponding to the j^{th} class and x being the input vector. The e function is utilized because it maps any value to a positive values which after which the vector is normalized. This yields us probabilities which must be positive (e handled that) and between 0 and 1 (normalization handled that). In our implementation, this translates to keeping track of a list of weights and running another for loop instead each iteration updating each weight individually.

3.3 The loss function

The loss function utilized is the cross entropy loss. This change was required because we now have to keep track of the loss for each class in our classification problem. Following this our implementation keeps track of a vector of losses the values of which are summed up and averaged at the end of each epoch. Mathematically, our loss function is:

$$Loss(a, b) = -\sum_{i=1}^{C} a_i \log b_i$$

where a is the actual distribution of a certain classification problem (in our case it is 0 for the wires that are not to be cut and 1 for the wire that is to be cut) and b is a distribution of the predictions generated by dotting the current weight vector with the input vector. Each element in the loss vector is summed up until the end of each training iteration and is then each element is averaged.

3.4 The training algorithm

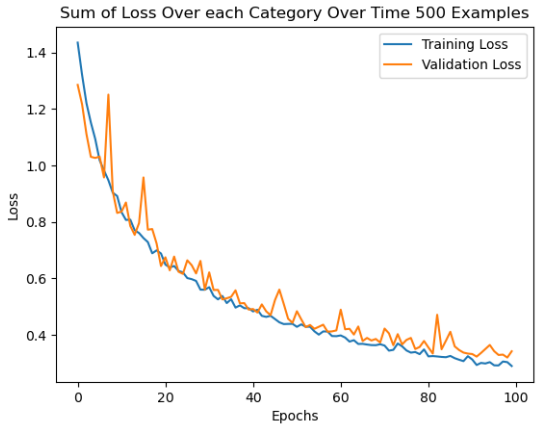
The details of the training algorithm used are basically the same as task 1 with two changes. Firstly, the prediction of a certain input is transformed by the e function (which is later normalized) instead of the sigmoid function. Secondly, since we are to compute a vector of weights, another for loop is run inside each iteration of training to update each weight individually. We also found from testing that a lower learning rate worked better for task 2.

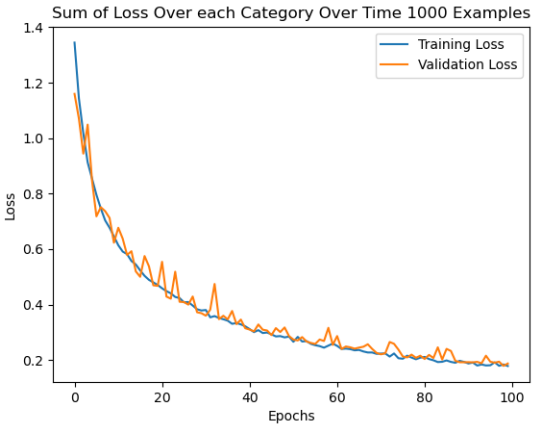
3.5 Dealing with overfitting

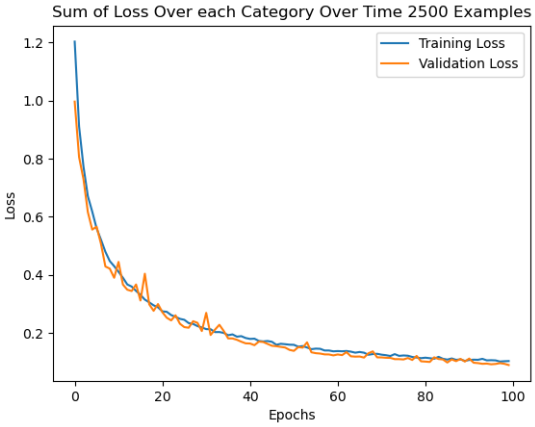
The techniques utilized to deal with overfitting were the same as task1 (data augmentation and hold out set for validation). Initially, we did see overfitting with this model but that was because our learning rate was not tuned properly. We did try to include L2 regularization at this point (simply adding $2 * \lambda * weight[i]$ to weight updates, where i is the the weight vector corresponding to the i^{th} class) but it did not seem to help much probably because lambda was not tuned properly. In the end, once we found a good learning rate, regularization was abandoned because our model was not overfitting (see data in the next section).

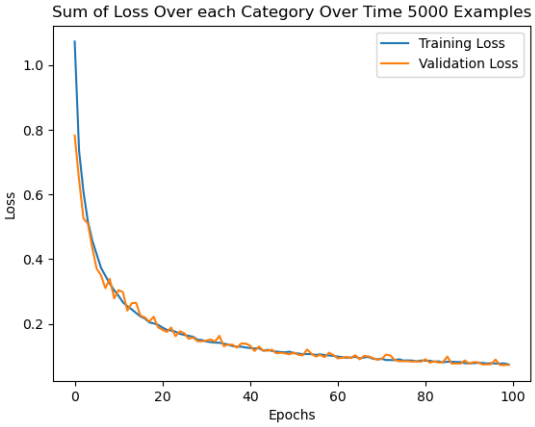
3.6 Data and analysis

All the data was collected over a learning rate of .0005 over 100 training epochs. The total data sample was broken down 90% for training and 10% for validation. Moreover, a separate data set of 100 samples was generated to test the accuracy of the models. The models do not demonstrate overfitting because the validation loss does not diverge from the training loss. Early stopping would be helpful in the case of overfitting and details of how it can be added to our current implementation have been included in the 'Dealing with overfitting' section of task1. The testing of the models showed promising results with the models correctly categorizing an image as dangerous or not 95% of the time when trained over 500 examples, 96% of the time when trained over 1000 examples, 99% of the time when trained over 2500 examples and 97% of the time when trained over 5000 examples. Data can be generated by running python3 Task2.py.









The models seem to capture the problem pretty well according to the data because validation loss stays close to the training loss and model predictions on a given input are highly accurate.

4 Credits

4.1 Arjun

1. All sections of the report
2. Tuning the features
3. Implemented softmax regression
4. Data augmentation optimization
5. Holdout set and validation loss implementation
6. Implemented the feature extraction part of task runner

4.2 Sai

1. Implemented generation of the wiring diagrams
2. Tuning and finding optimal learning rates
3. Implemented logistic regression
4. Tried and implemented regularization which was abandoned later
5. Implemented task runners