Principles of Machine Learning (CS4011)

Programming Assignment 1B

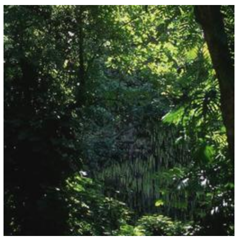
Ameet Deshpande - (CS15B001)

September 12, 2017

1 Question 7 - Logistic Regression

1.1 Understanding the data

- The data given is in the form of images. It has 4 classes. The training and test examples have already been split. There are approximately 80 training examples and 1000 test examples. This gives a test-train split of about 8%.
- Feature extraction is a very important part of Machine Learning pipeline. For doing feature extraction, code has been borrowed from Professor Boyd's files which already extract features after going through all the directories.
- Observing the feature extraction procedure, it can be seen that a very naive way is being used to do the same. 32 bins representing pixel intensities between $0 - 256$ are created. The same is done for all 3 colours. This thus results in $32 \times 3 = 96$ features. No information about edges or lines are used. It uses a bag of pixel model where no spacial correlation is calculated. But as we will see later, this model does very well to classify. This is because, the two classes considered, Mountain and Forest, typically have very different colours. Observing a few images from the training dataset, we can see that Forest images are predominatly green and mountain images are predominantly white and blue. But as shown below, there are ambiguous pictures, but the code still ends up performing very well.


(a) Forest

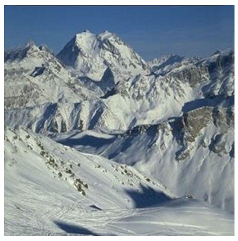

(b) Mountain

Figure 1: Two typical figures from the dataset

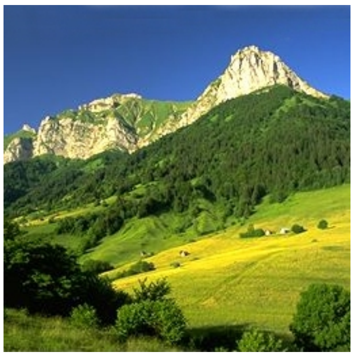


Figure 2: Ambiguous Example

- Viewing the CSV files shows that values range from 0 up till 6000. This will cause problems during training the data. The Data is thus scaled and normalized to get it in a smaller range to work with. This will also later on help when we are training the neural networks.

1.2 Training a Logistic Classifier

- In this part we are supposed to train our own Logistic Classifier. I have purposely set the regularization parameter as 0 as this will serve as a comparison between the regularized and non-regularized versions of Logistic Regression.
- The function from *sklearn* is used to do the same. The value of parameter is set as $C = 10^8$. $C = \frac{1}{\lambda}$ and thus, this means that the regularization parameter is set to be very small. The logistic is trained on the csv file which contains the normalized data. The maximum number of iterations is 100. The logistic converges well before that (in about 31 iterations). The following are the accuracy scores that were obtained.

Class	Precision	Recall	F-Score
Class 1	0.957	0.960	0.958
Class 2	0.953	0.949	0.951

- As can be seen, this is pretty good performance for a baseline model. These values can now be used to compare against other values. The number of examples in each class is almost the same and thus the precision and recall values are a good indicator of the average as well, which is pretty good. The values are also tabulated when the script is run, for convenience.

1.3 L1 Logistic Regression

- Professor Boyd's code uses L1 logistic regression to classify. As as have studied in class, L1 logistic regression will prevent over fitting by driving weights to 0.

Though this image data is not very big, in general it helps to run L1 regularization as it will lead to sparse matrices and thus efficient running. The code is mainly written in C and uses the efficient Linear Algebra libraries like *BLAS* and *LAPACK*.

- Three main functions are provided, them being l1_logreg_train for training l1_logreg_classify for classification l1_logreg_regpath for (approximate) regularization path computation. Bash commands are used to execute this model. We use the *os.system* call for executing the bash commands from within python.
- The predicted labels are stored in a different file format. The file is read and the predicted labels are retrieved to evaluate a few metrics.
- The code recommends 0.01 as the regularization parameter for the given dataset. But we run the model with different regularization parameters to check which gives the best performance. The following is the graph for the precisions v/s Lambdas. Note that the precisions are the average of the precisions of the two classes. This has been used as the metric to decide what regularization parameter to use.

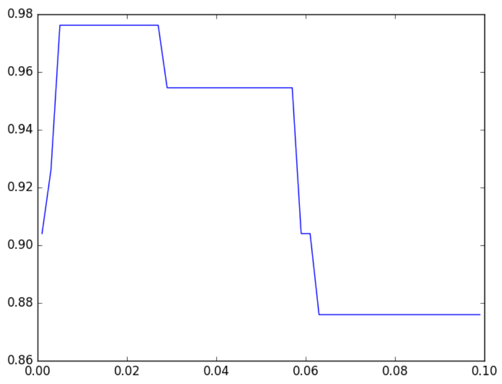


Figure 3: Precision v/s. Regularization Parameter

- From the above figure it can be seen that regularization parameter between $0.007, 0.27$, perform almost the same. We thus use 0.01 as an optimal value as suggested by the Professor. The following table summarizes all the experiments. The regularized code performs better than the non-regularized code.

Model	Class	Precision	Recall	F-Score
Logistic	Class 1	0.957	0.960	0.958
Logistic	Class 2	0.953	0.949	0.951
Boyd	Class 1	1.000	0.95	0.9743
Boyd	Class 2	0.9523	1.000	0.975

2 Question 8 - Backpropagation

2.1 Original Backpropagation Algorithm

- We implement the backpropagation algorithm using standard error functions. We however need to define an error function that can be used. We use Cross-Entropy loss and justify the same below.

- We have two options that we are looking at. One is the Cross-Entropy loss and the other is the Mean Squared Error. Both measure the closeness of the output compared to actual values. It is assumed that a *softmax* was run in the last layer to predict probabilities. Mean squared error is known to give emphasis on incorrect examples. Even if one model is doing better than the other overall, if it is making mistakes where the predicted values are very bad, then MSE will favour the second model, which should not be the case.

- The Neural Network has just one hidden Layer. The first layer is an input layer. The second layer is the hidden layer and uses *tanh* sigmoidal activation function. As seen in class, this sigmoid is better than the logistic as it has more degrees of freedom. Technicaly a ReLU should be used, but since the network is not very deep, we do not face the problem of vanishing gradients if we initialize the weights to be small and close to 0. Below is a diagram of what the architecture approximately looks like.

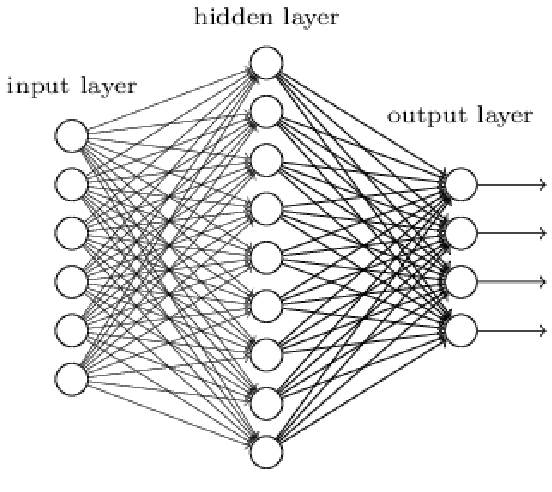


Figure 4: Network Architecture

- Few words about the design of the code. The code defines classes to handle neural networks. There are train and predict functions which assist in encapsulating the main functionalities.

2.2 Derivation for Cross-Entropy Loss

- Note that all the derivations assume a single training examples. This can be extended to the full batch case very easily and is implemented directly in the code.
- The notation that will be used in the rest of the report is the following. Hidden layer is referred to as layer 1 and output layer is layer 2.
  - X - Input data to layer 1
  - y denotes the output
  - a^1, a^2 - Input activations of layer 1 and 2
  - h^1, h^2 - Output activations from layer 1 and 2
  - W^1, W^2 - Weight matrices in both layers
  - $\hat{y}$ is the output of the network and equals h^2.
  - $\mathcal{L}$ denotes the Loss function.
  - l denotes the true label
  - $\circ$ represents elementwise multiplication
  - The bias terms are purposely omitted as they can be added as a row to the matrix and a column of $1's$ can be appended to the data matrices.
- The derivation is given below. Vector representation is used everywhere and only one training example is considered.

$$\mathcal{L} = -\sum_{i=1}^{n} y_i \times \log \hat{y}_i \tag{1}$$

$$\frac{\partial \mathcal{L}}{\partial h_i^2} = -\frac{y_i}{\hat{y}_l} \tag{2}$$

$$\frac{\partial \mathcal{L}}{\partial h^2} = -\frac{y}{\hat{y}_l} \tag{3}$$

$$\frac{\partial \mathcal{L}}{\partial a_i^2} = \frac{\partial \mathcal{L}}{\partial h_j^2} \times \frac{\partial h_j^2}{\partial a_i^2} \tag{4}$$

$$\frac{\partial \mathcal{L}}{\partial a_i^2} = \frac{\partial(-\log \hat{y}_l)}{\partial a_i^2}, \text{ for one training example} \tag{5}$$

$$\implies \frac{\partial \mathcal{L}}{\partial a_i^2} = \frac{-1}{\hat{y}_l} \times \frac{\partial softmax(a^2)_l}{\partial a_i^2} \tag{6}$$

$$\implies \frac{\partial \mathcal{L}}{\partial a_i^2} = \frac{-1}{\hat{y}_l} \times \left(\frac{\frac{\partial e^{a_l^2}}{\partial a_i^2}}{\sum_i' e^{a_{i'}^2}} - \frac{e^{a_l^2} \times \left(\frac{\partial \sum_i' e^{a_i^2}}{\partial a_i^2} \right)}{\left(\sum e^{a_{i'}^2} \right)^2} \right) \tag{7}$$

$$\implies \frac{\partial \mathcal{L}}{\partial a_i^2} = \frac{-1}{\hat{y}_l} \times \left(\mathbb{1}_l \times softmax(a^2)_l - softmax(a^2)_l \times softmax(a^2)_i\right) \tag{8}$$

$$\implies \frac{\partial \mathcal{L}}{\partial a_i^2} = \frac{-1}{\hat{y}_l} \times \left(\mathbb{1}_l \times \hat{y}_l - \hat{y}_l \times softmax(a^2)_i\right), \mathbb{1}_l \text{ is 0 for } i \neq l \tag{9}$$

$$\implies \frac{\partial \mathcal{L}}{\partial a_i^2} = \frac{-1}{\cancel{\hat{y}_l}} \times \left(\mathbb{1}_l \times \cancel{\hat{y}_l} - \cancel{\hat{y}_l} \times softmax(a^2)_i\right) \tag{10}$$

$$\implies \frac{\partial \mathcal{L}}{\partial a^2} = -(y - \hat{y}) \tag{11}$$

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^2} = \frac{\partial \mathcal{L}}{\partial a_j^2} \times \frac{\partial a_j^2}{\partial W_{ij}^2} \tag{12}$$

$$\frac{\partial a_j^2}{\partial W_{ij}^2} = h_i^1 \tag{13}$$

$$\implies \frac{\partial \mathcal{L}}{\partial W_{ij}^2} = h_i^1 \times \frac{\partial \mathcal{L}}{\partial a_j^2} \tag{14}$$

$$\implies \frac{\partial \mathcal{L}}{\partial W_{ij}^2} = h_i^1 \times \frac{\partial \mathcal{L}}{\partial a^2{}_j}, \text{ using matrix element} \tag{15}$$

$$\implies \frac{\partial \mathcal{L}}{\partial W^2} = \left(h^1\right)^T \times \frac{\partial \mathcal{L}}{\partial a^2}, \text{ writing it in matrix form} \tag{16}$$

$$\frac{\partial \mathcal{L}}{\partial h_i^1} = \sum_j \frac{\partial \mathcal{L}}{\partial a_j^2} \times \frac{\partial a_j^2}{\partial h_i^1} \tag{17}$$

$$\frac{\partial \mathcal{L}}{\partial h_i^1} = \sum_j \frac{\partial \mathcal{L}}{\partial a_j^2} \times W_{ij}^2 \tag{18}$$

$$\implies \frac{\partial \mathcal{L}}{\partial h^1} = \frac{\partial \mathcal{L}}{\partial a^2} \times W^{2^T}, \text{ writing it in matrix form} \tag{19}$$

$$\frac{\partial \mathcal{L}}{\partial a_i^1} = \frac{\partial \mathcal{L}}{\partial h_i^1} \times \left(1 - tanh^2(a_i^1)\right) \tag{20}$$

$$\implies \frac{\partial \mathcal{L}}{\partial a^1} = \frac{\partial \mathcal{L}}{\partial h^1} \circ \left(1 - tanh^2(a^1)\right), \circ \text{ represents element wise multiplication} \tag{21}$$

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^1} = \frac{\partial \mathcal{L}}{\partial a_j^1} \times \frac{\partial a_j^1}{\partial W_{ij}^1}, \text{ using equations similar to } 14, 15 \tag{22}$$

$$\frac{\partial a_j^1}{\partial W_{ij}^1} = X_i \tag{23}$$

$$\implies \frac{\partial \mathcal{L}}{\partial W_{ij}^1} = X_i \times a_j^1 \tag{24}$$

$$\implies \frac{\partial \mathcal{L}}{\partial W^1} = X^T \times \frac{\partial \mathcal{L}}{\partial a^1}, \text{ similar to equation 16} \tag{25}$$

2.3 Derivation for defined Loss function

- Notation used in this subsection is the same as the previous subsection. For simplicity only one training example is considered here and it can easily be extended to multiple examples, as done in the code.

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{K}(y_{ik} - f_k(x_i))^2 + \gamma(\sum_k\sum_m \beta_{km}^2 + \sum_m\sum_l \alpha_{ml}^2) \tag{26}$$

Note that we will be considering the case when $N = 1$

f_k is the same as h^2 from the notation above

$$\frac{\partial\mathcal{L}}{\partial h_i^2} = h_i^2 - y_i, \text{ Note that the } \alpha \text{ and } \beta \text{ denote weights and thus derivatives are 0} \tag{27}$$

$$\frac{\partial\mathcal{L}}{\partial h^2} = h^2 - y, \text{ in vector notation} \tag{28}$$

$$\frac{\partial\mathcal{L}}{\partial a_i^2} = \sum_j \frac{\partial\mathcal{L}}{\partial h_j^2} \times \frac{\partial h_j^2}{\partial a_i^2} \tag{29}$$

$$\frac{\partial h_i^2}{\partial a_i^2} = -(h_i^2 \times h_j^2)\forall i \neq j \tag{30}$$

$$\frac{\partial h_j^2}{\partial a_i^2} = (h_i^2 - h_i^2 \times h_j^2)\forall i = j \tag{31}$$

$$\frac{\partial\mathcal{L}}{\partial a_i^2} = (h_j^2 - y_j) \times -(h_i^2 \times h_j^2)\forall i \neq j \tag{32}$$

$$\frac{\partial\mathcal{L}}{\partial a_i^2} = (h_j^2 - y_j) \times (h_i^2 - h_i^2 \times h_j^2)\forall i = j \tag{33}$$

Clubbing equations 30 and 31 (34)

$$\frac{\partial h_j^2}{\partial a_i^2} = h_i^2 \times \left((h_i^2 - y_i) - \sum_j (h_j^2 - y_j) \times h_j^2\right) \tag{35}$$

A vector minus number is assumed to mean element wise subtraction

Converting the above to vector notation, we have

Please note that W^2 does not denote $(W)^2$

$$\frac{\partial\mathcal{L}}{\partial a^2} = h^2 \circ \left((h^2 - y) - (h^2 - y) \circ h^2\right) \tag{36}$$

$$\frac{\partial\mathcal{L}}{\partial W_{ij}^2} = \frac{\partial\mathcal{L}}{\partial a_j^2} \times \frac{\partial a_j^2}{\partial W_{ij}^2} + 2\gamma \times W_{ij}^2 \tag{37}$$

$$\frac{\partial a_j^2}{\partial W_{ij}^2} = h_i^1 \tag{38}$$

$$\implies \frac{\partial\mathcal{L}}{\partial W_{ij}^2} = h_i^1 \times \frac{\partial\mathcal{L}}{\partial a_j^2} + 2\gamma \times W_{ij}^2 \tag{39}$$

$$\implies \frac{\partial\mathcal{L}}{\partial W_{ij}^2} = h_i^1 \times \frac{\partial\mathcal{L}}{\partial {a^2}_j}, \text{ using matrix element} \tag{40}$$

$$\implies \frac{\partial \mathcal{L}}{\partial W^2} = \left(h^1\right)^T \times \frac{\partial \mathcal{L}}{\partial a^2} + 2\gamma \times W^2, \text{ writing it in matrix form} \tag{41}$$

$$\frac{\partial \mathcal{L}}{\partial h_i^1} = \sum_j \frac{\partial \mathcal{L}}{\partial a_j^2} \times \frac{\partial a_j^2}{\partial h_i^1} \tag{42}$$

$$\frac{\partial \mathcal{L}}{\partial h_i^1} = \sum_j \frac{\partial \mathcal{L}}{\partial a_j^2} \times W_{ij}^2 \tag{43}$$

$$\implies \frac{\partial \mathcal{L}}{\partial h^1} = \frac{\partial \mathcal{L}}{\partial a^2} \times W^{2^T}, \text{ writing it in matrix form} \tag{44}$$

$$\frac{\partial \mathcal{L}}{\partial a_i^1} = \frac{\partial \mathcal{L}}{\partial h_i^1} \times \left(1 - tanh^2(a_i^1)\right) \tag{45}$$

$$\implies \frac{\partial \mathcal{L}}{\partial a^1} = \frac{\partial \mathcal{L}}{\partial h^1} \circ \left(1 - tanh^2(a^1)\right), \circ \text{ represents element wise multiplication} \tag{46}$$

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^1} = \frac{\partial \mathcal{L}}{\partial a_j^1} \times \frac{\partial a_j^1}{\partial W_{ij}^1} + 2\gamma \times W_{ij}^1 \tag{47}$$

$$\frac{\partial a_j^1}{\partial W_{ij}^1} = X_i \tag{48}$$

$$\implies \frac{\partial \mathcal{L}}{\partial W_{ij}^1} = X_i \times a_j^1 + 2\gamma \times W_{ij}^1 \tag{49}$$

$$\implies \frac{\partial \mathcal{L}}{\partial W^1} = X^T \times \frac{\partial \mathcal{L}}{\partial a^1} + 2\gamma \times W^1 \tag{50}$$

2.4 Evaluation with Cross Entropy Loss

- One hyper-parameter of the model is the number of units in the hidden layer. After using Cross-Entropy loss, 45 hidden layers seems to be a good option, with the average of F-scores of all 4 classes as the metric. Thus 45 hidden layers are used throughout to compare different models and parameters. Technically, we should be using a Cross-Validation set to perform comparisons related to hyper-parameters. But in this dataset, the size is very small and we cannot afford to lose any more training examples when we are training a Deep-Neural Network. The training error continuously decreases as the training progresses, but the test error decreases and then increases. We set the number of iterations as 1000, as over-fitting occurs if the number of iterations increases.

Class	Precision	Recall	F-Score
Class 1	0.5	0.5	0.5
Class 2	0.789	0.75	0.769
Class 3	0.565	0.65	0.60
Class 4	0.555	0.5	0.526

- The learning rate that is used is 0.01. This is chosen as the error continually decreases with this learning rate and the convergence is also not slow.

Learning Rate	Hidden Units	Iterations
0.01	45	1000

- The number of hidden units can actually be further reduced as the features are just colours. There should intuitively be no more than 10-15 combinations.

2.5 Evaluation with modified Loss function

- The hyperparameters are kept the same as last time as this will facilitate ease in comparison. We expect the regularized parameter to do better than the previous model as we expect fewer features to govern the model. Regularizing helps get some weights to 0.
- The following graph and table show how the model performed for different regularization parameters. The best scores are obtained with the regularization parameter of 1

γ	Class	Precision	Recall	F-Score
0.01	Class1	0.55	0.5	0.526
0.01	Class2	0.6	0.75	0.666
0.01	Class3	0.60	0.7	0.65
0.01	Class4	0.57	0.4	0.47
0.1	Class1	0.35	0.25	0.29
0.1	Class2	0.72	0.8	0.76
0.1	Class3	0.5	0.6	0.545
0.1	Class4	0.5	0.5	0.5
1	Class1	0.75	0.6	0.66
1	Class2	0.73	0.7	0.71
1	Class3	0.65	0.85	0.73
1	Class4	0.68	0.65	0.66
10	Class1	0.52	0.65	0.57
10	Class2	0.5	0.8	0.61
10	Class3	0	0	0
10	Class4	0.318	0.35	0.33
100	Class1	0.125	0.35	0.1842
100	Class2	0	0	0
100	Class3	0	0	0
100	Class4	0	0	0

- To check what is happening to the weights, we calculate $sum(W^1 \circ W^1)$ and $sum(W^2 \circ W^2)$. This gives us a good idea of the magnitude of the weights. The following are the values for matrix 1. As can be seen, the magnitude of the weights overall decreases considerably. This is because of the regularization

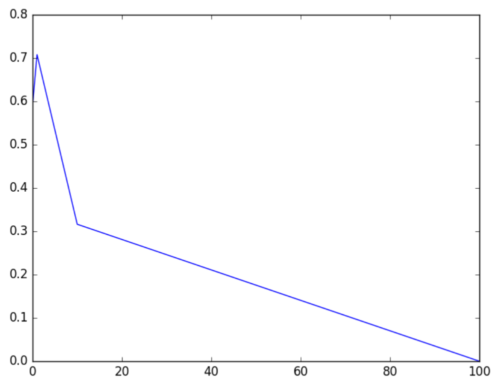


Figure 5: FScore v/s. γ (Notice the peak)

term that forces the weights to be small. It can thus be inferred that regularization successfully reduces the complexity of the model. The regularized model with $\gamma = 1$ performs the best and its performance across all classes is very even. This shows that over-fitting has been avoided. Thus, even though we had a high number of units in the hidden layer, we have avoided over-fitting. Also, printing the weight matrices shows that the weights have actually reduced considerably. But unlike Lasso Regression, the weights were not driven to 0. Weights were driven to values as small as 0.005. This shows that those features are unnecessary and that we could probably have done as well with a less complex model.

γ	$W^1 \circ W^1$
0.01	208.30
0.1	121.21
1	64.06
10	5.859