Project 2: Handwriting Detection using Linear and Logistic Regression and Neural Network

Abhav Luthra
Graduate student, Computer Science
#Person 50288904
abhavlut@buffalo.edu

1. Description of Model and Procedure followed

- **Abstract of Problem and Dataset**
  In this project we are applying the concept of linear regression, Logistic regression and Neural Network on CEDAR Letter dataset.

  Our dataset uses “AND" images samples extracted from CEDAR Letter dataset. The CEDAR dataset consists of handwritten letter manuscripts written by 1567 writers. Each of the writer has copied a source document thrice. Hence, there are a total of 4701 handwritten letter manuscripts. Each letter has vocabulary size of 156 words (32 duplicate words, 124 unique words) and has one to five occurrences of the word ‘AND’ (cursive and hand printed). Image snippets of the word “AND" were extracted from each of the manuscript using transcript-mapping function of CEDAR-FOX.

  The total number of “AND" image fragments available after extraction are 15,518.

  The features are obtained from two different sources:
  1. Human Observed features: Features entered by human document examiners manually
  2. GSC features: Features extracted using Gradient Structural Concavity (GSC) algorithm.

  Human observed features have 9 features and GSC have 512 features. We also have files with same author images id mapped together and different authors images id mapped together. All the features are initially normalized to fall in the interval of [0, 1].

- **Splitting and cleaning the Data Set**
  Given data for same and different pair of authors is unequal in number and therefore to train our model we randomly put **equal no of same and different pair and join the features** to them.

  Features of the two images can be **concatenated** to form an 18-column matrix or **subtracted** to form a 9-column matrix for human observed dataset. Then the data is **cleaned** i.e. the column with zero variance are removed and then the **data is split** into 80% for data training 10% for validation and 10% for testing.

  Zero Variance means that whole column has the same value therefore it makes no contribution in linear regression. It also doesn’t allow us to calculate inverse in Phi matrix.

- **Regression Model**
  Linear regression is a **linear** approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables) that are continuous in nature. We have a list of 9 independent variable also called features above and a target variable that is computed using basis function. This problem categorically falls in classification as it gives output in form of class 0 and 1. But we convert our linear output to class by rounding it to nearest integer. The problem here is solved by gradient descent.

Linear Regression function y(x,w) has the form,

$$T=W^T\Phi(x)$$

where w = (w0;w1;wM-1) is a weight vector to be learnt from training samples and Φ = (Φ1, Φ2, …) is a vector of M basis functions

In this project we use Gaussian radial basis function

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^\top \Sigma_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right)$$

where μ_j = Centroid of each cluster
$\sum^{-1}_j$ = Variance - Covariance Matrix

- **Stochastic Gradient Descent Solution**
  In Gradient Descent optimization, we compute the cost gradient based on the complete training set; hence, we sometimes also call it *batch Gradient Descent*. In case of very large datasets, using Gradient Descent can be quite costly since we are only taking a single step for one pass over the training set -- thus, the larger the training set, the slower our algorithm updates the weights and
  the longer it may take until it converges to the global cost minimum.

  The term "stochastic" comes from the fact that the gradient based on a single training sample is a "stochastic approximation" of the "true" cost gradient. Due to its stochastic nature, the path towards the global cost minimum is not "direct" as in GD, but may go "zig-zag" if we are visualizing the cost surface in a 2D space. However, it has been shown that SGD almost surely converges to the global cost minimum if the cost function is convex
  Weights are updated in this form:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)}$$

W is calculated by first order linear differentiation of weights given by:

$$\nabla E = \nabla E_D + \lambda \nabla E_W$$

$$\nabla E_D = -(t_n - \mathbf{w}^{(\tau)\top}\phi(\mathbf{x}_n))\phi(\mathbf{x}_n)$$
$$\nabla E_W = \mathbf{w}^{(\tau)}$$

Where lambda is the learning rate for the optimization function
Now that the model is trained, we can find the error in the model. We also need to add a regularization term to the error function,
$E(w) = E_D(w) + \boldsymbol{\lambda} E_w(w)$

Where $\boldsymbol{\lambda}$ is the relative importance of the regularization term and

$$E_w(w) = 0.5\, w^T w$$

And E is the root mean square of form,

$$E_{RMS} = \sqrt{}2E(w^*)/N_V$$

Where E(w) is of form,

$$E_D = 0.5 \sum (t_n - w^T \Phi(x_n))^2$$

This is the root mean square, it is reduced to optimize our model.

- **<u>Logistic Model</u>**

  Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is binary in nature like in this project. It is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

  We use our old linear regression algorithm to try to predict y given x. Howeve, $h_\theta(x)$ to take values larger than 1 or smaller than 0 i.e. $y \in \{0, 1\}$. Our hypotheses is $h_\theta(x)$ to satisfy $0 \le h_\theta(x) \le 1$. This is accomplished by plugging $\theta^T x$ into the Logistic Function.

  Logistic Model uses the "Sigmoid Function," also called the "Logistic Function":

$$h_\theta(x) = g(\theta^T x)$$

$$z = \theta^T x$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

  In Logistic Regression, we use the same gradient descent approach, here the cost function is as follows:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\mathrm{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if } y = 1$$

$$\mathrm{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if } y = 0$$

  Vectorized form required in this project is:

$$h = g(X\theta)$$

$$J(\theta) = \frac{1}{m} \cdot \left(-y^T \log(h) - (1 - y)^T \log(1 - h)\right)$$

  Regularized Form used:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Regularized cost function is used in this project even though there is not much difference observed between regularized cost function and non-regularized cost function.

The λ, or lambda, is the **regularization parameter**. It determines how much the costs of our theta parameters are inflated.

Cost function is minimized using the scholastic gradient descent model explained above in linear regression.

- **Neural Network Model**

  Artificial neural networks (ANN) are computing systems vaguely inspired by the biological neural networks that constitute animal brains. The neural network itself isn't an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

  The basic unit of computation in a neural network is called a neuron or a node or perceptron. It receives input from some other nodes, or from an external source and computes an output.
  Most simple Neural Network have generally 3 layers of neurons, each layer can have multiple neuron. In a 3 layer network, first layer is called input layer, second is hidden and third is output layer. N-layer Neural Network have N-2 hidden layers. Our model is simple and therefore, have 3 layers only.

  The activation function used on the dense layer is 'Relu'. The choice of this activation function is to make the neurons less sparse and the computation less costly. The activation function used for the final layer is 'Softmax'. This is used as the 'softmax' function gives the categorical probability distribution of the output. Hence, a higher probability may suggest that it belongs to that particular class. We also add dropout to the model to prevent overfitting of data. We use 'categorical_crossentropy' loss function and 'rmsprop' as an optimizer. Further, we use early stopping to stop the training if there is no improvement while training.

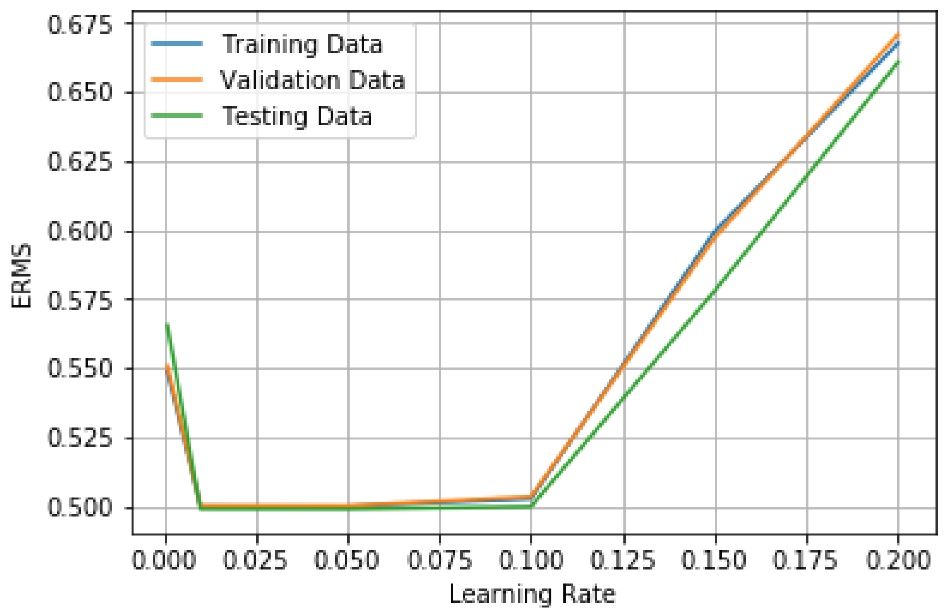
2. Experiment

   a. Human Observed DataSet

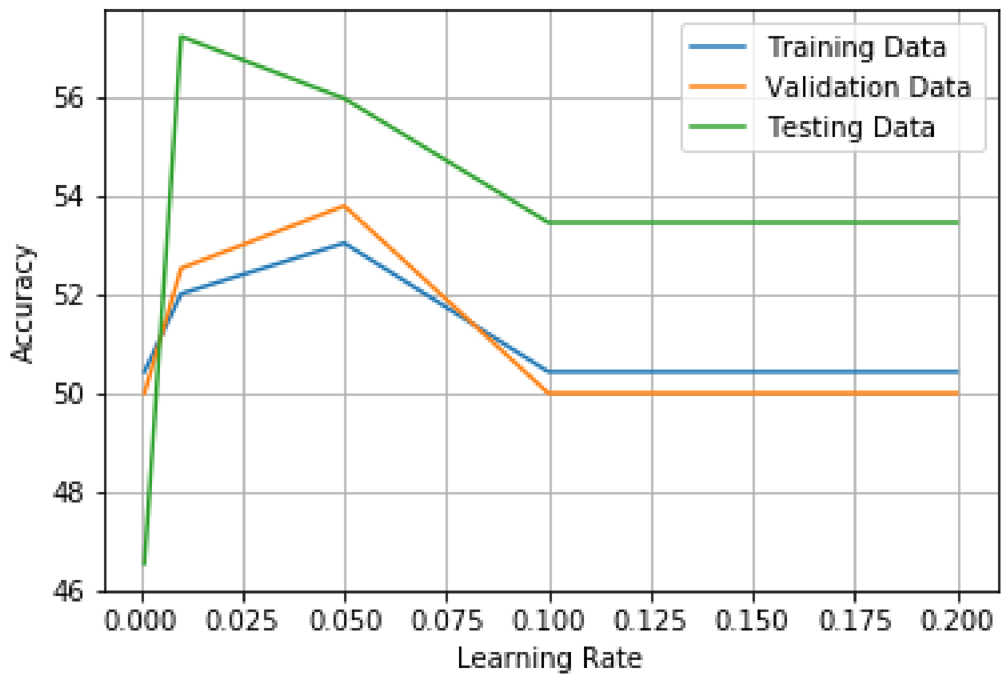
      i) Subtracted Data
         1. Linear Regression

Cluster Size is chosen as 6, to caputure variance even though the output remain same for cluster size of 4&5.

ERMS remain same for 0.001 – 0.1 and shoots up afterward. Full data set is considered while modelling the data.



Accuracy is at peak for validation and training data at **0.05 learning rate** and drops in both direction and flattens from 0.1-0.2 and decrease afterwards.



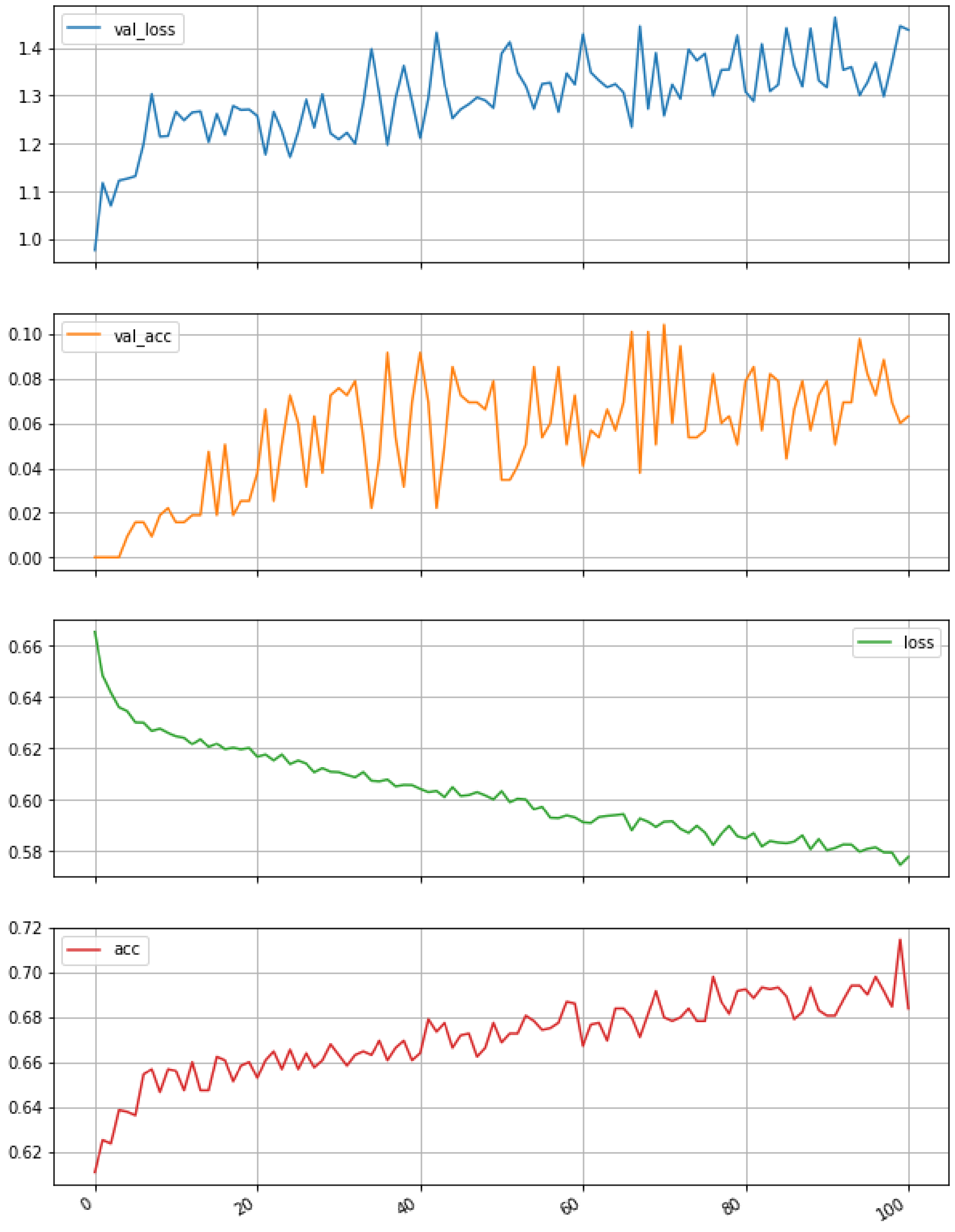
2. Logistic regression

Accuracy is calculated using the scipy.optimize library minimize function is observed as follows:

Accuracy for training = 49.56521739130435%
Accuracy for testing = 53.459119496855344%
Accuracy for validation = 50.0%

This low accuracy could be because of the small dataset available for model building.

3. Neural Network



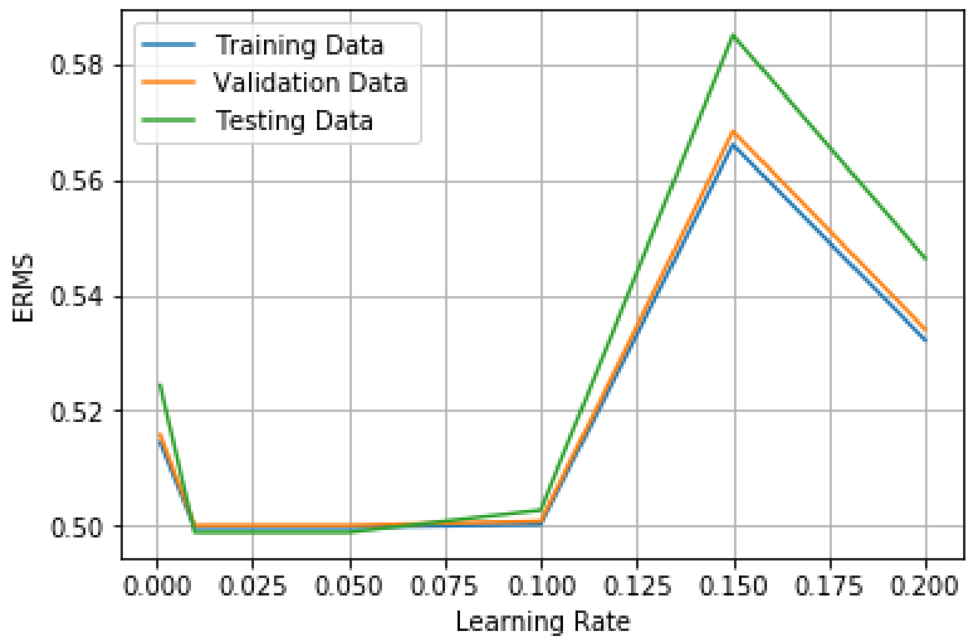
Testing Accuracy of 89.5 – 92% is observed for dense layer of 256. It can be further improved by increasing layer size but that will cause overfitting as data records for data modelling is very less.

Validation and training accuracy increases with epochs and error/ loss decreases which is ideal as shown in graph.

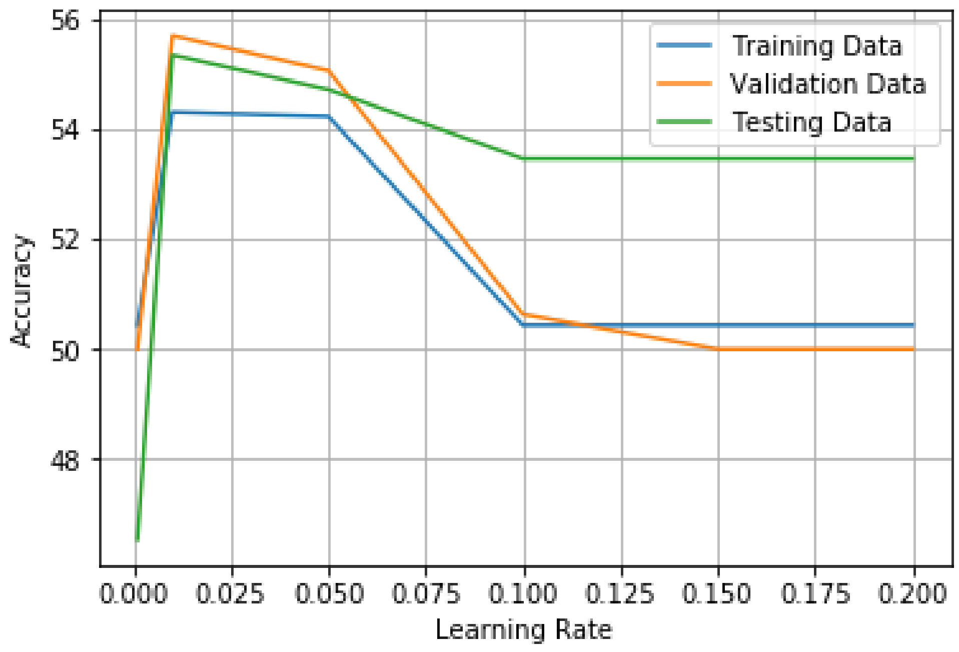
ii) Concatenated Data

1. Linear Regression

Cluster Size is chosen as 6, to capture variance accuracy increases while erms remain similar for 4&5.



ERMS remain same for 0.001 – 0.1 and shoots up afterward. Full data set is considered while modelling the data.

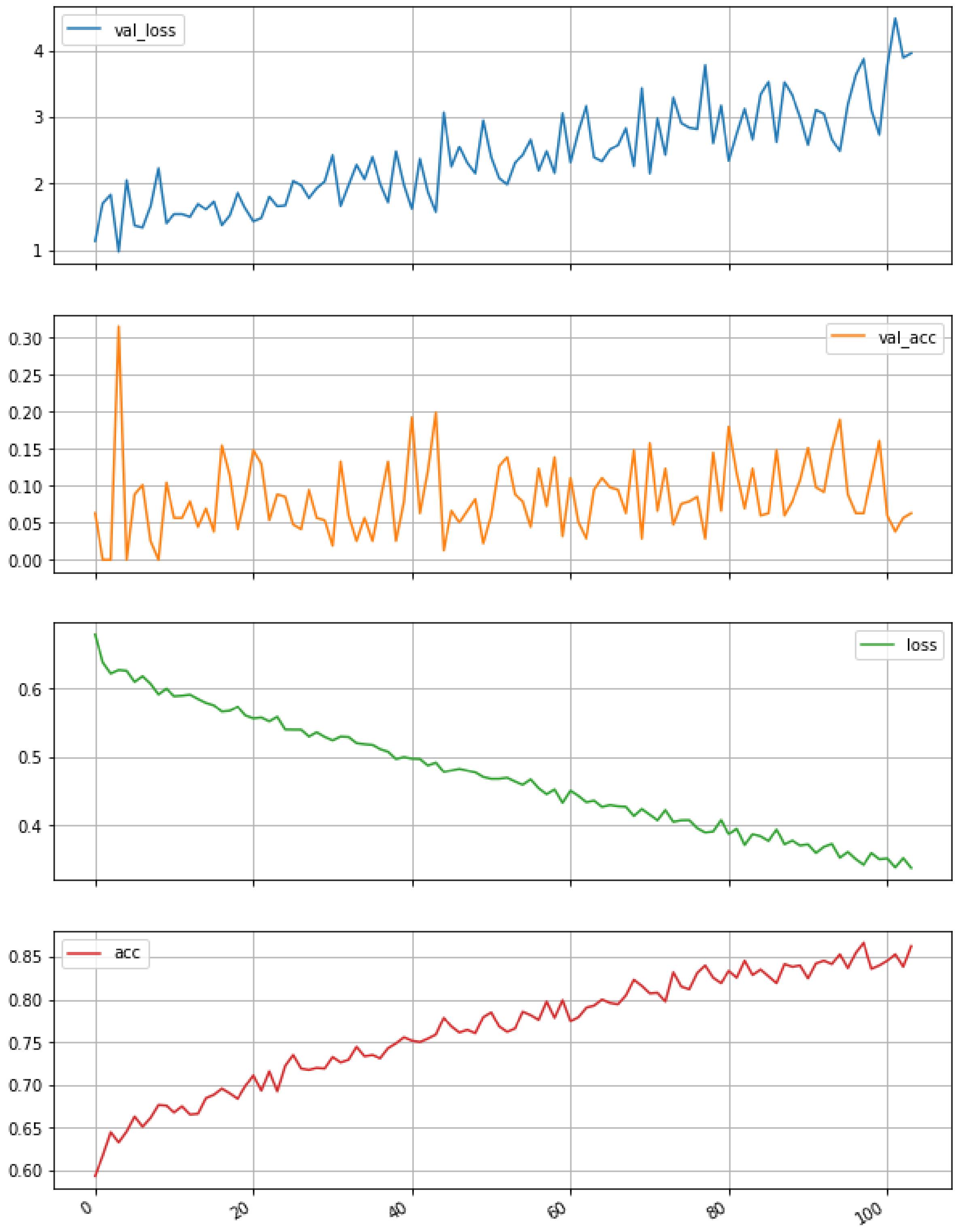


Accuracy is at peak for validation, testing and training data at **0.01 - 0.05 learning rate** and drops in both direction and flattens from 0.1-0.2 and decrease afterwards.

2. Logistic regression

Accuracy is calculated using the scipy.optimize library minimize function is observed as follows:
Accuracy training = 49.5%, Accuracy testing = 53.45%, Accuracy validation = 50.0%
This low accuracy could be because of the small dataset available for model building.

3. Neural Network



Testing Accuracy of 78.5 – 82% is observed for dense layer of 512. It can be further improved by increasing layer size but that will cause overfitting as data records for data modelling is very less.

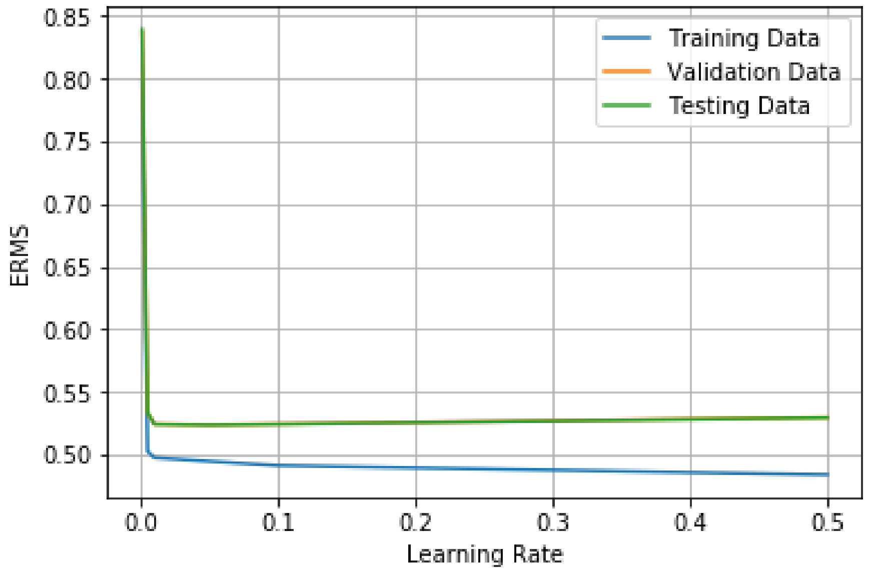
Validation and training accuracy increases with epochs and error/ loss decreases which is ideal as shown in graph. Though validation accuracy have a lot of noise and does not form a smooth graph.

b. GSC DataSet

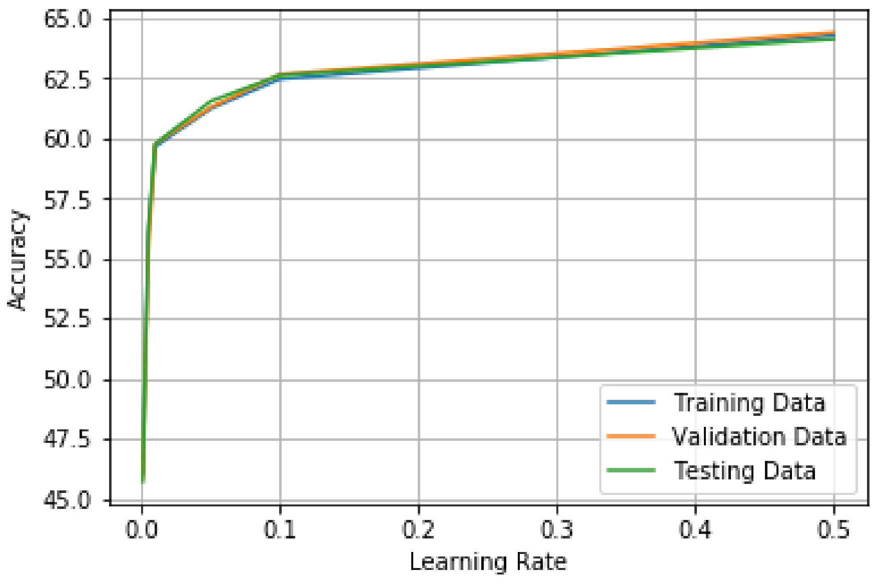
i) Subtracted Data

1. Linear Regression

Cluster Size is chosen as 10, to capture variance of the dataset.



ERMS remain decreasing for 0.001 – 0.5 for training data but slightly increases for testing and therefore is not further increased.

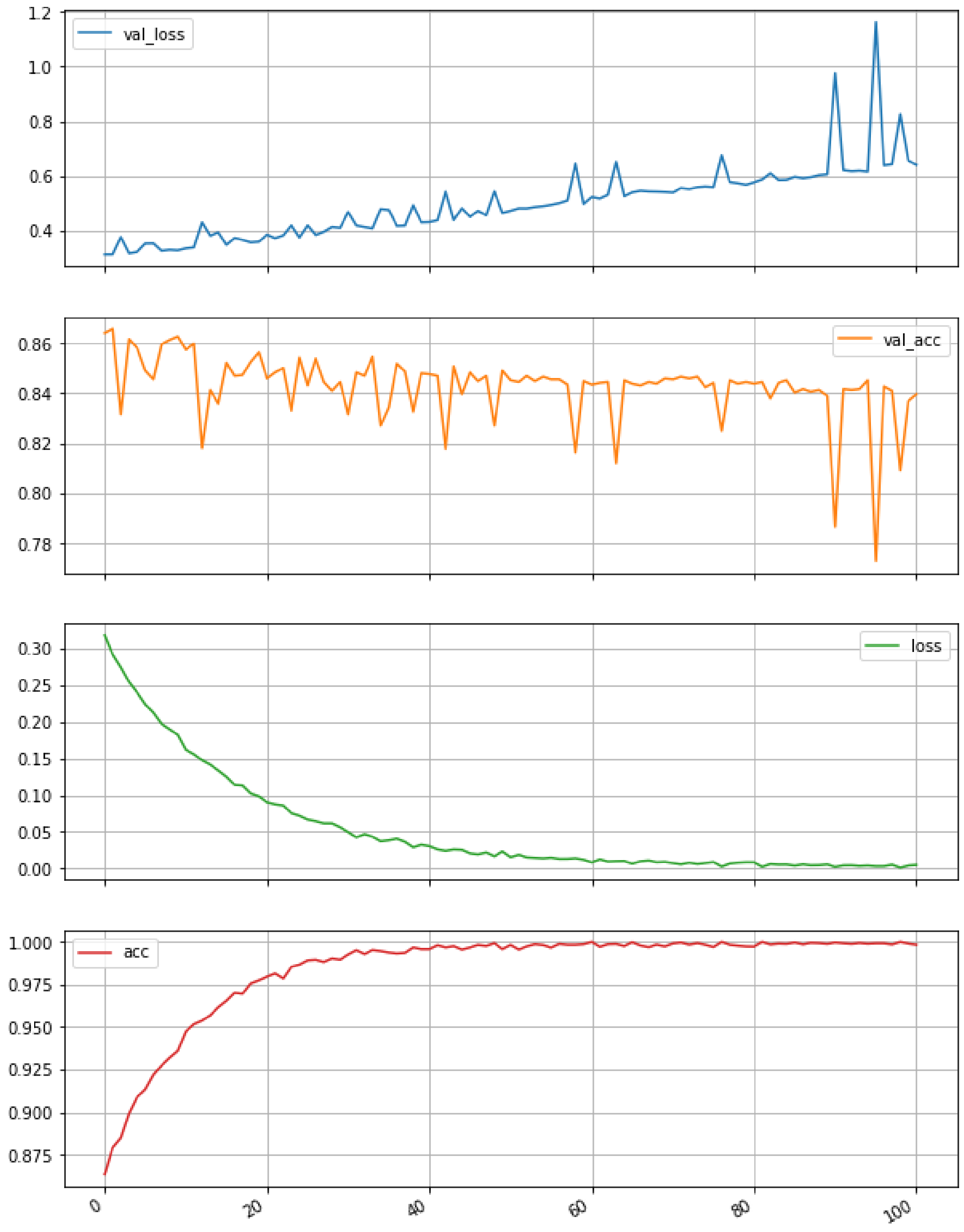


Accuracy is at peak for validation, testing and training data at **0.05 learning rate** and keeps on increasing from 0.01 – 0.05. Further values are not tried because of increases erms for testing data

2. Logistic regression

Accuracy is calculated using the scipy.optimize library minimize function is observed as follows:
Accuracy training = 48.3%, Accuracy testing = 49.53%, Accuracy validation = 49.65%
This low accuracy could be because of the small dataset used for model building.

3. Neural Network



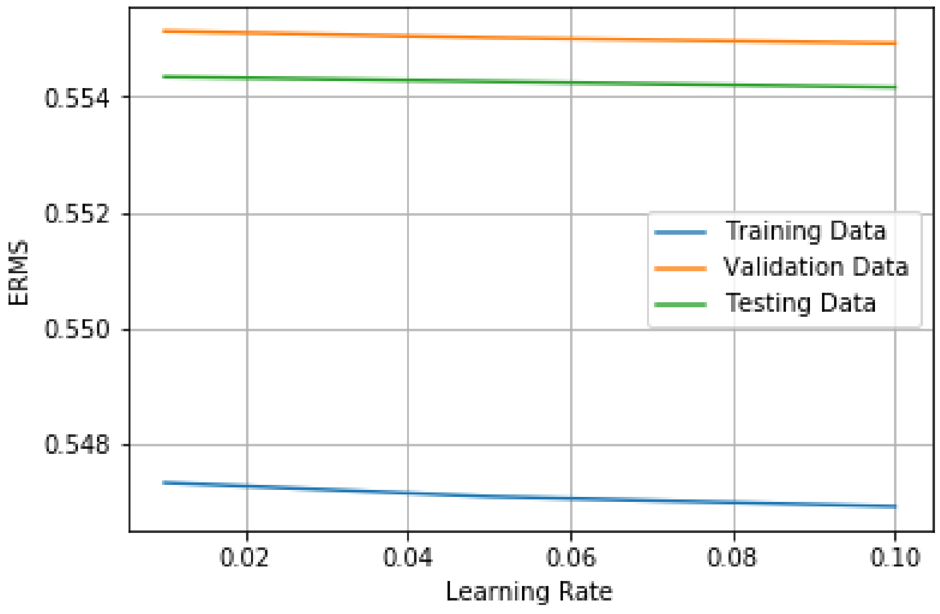
Testing Accuracy of 99%+ is observed for dense layer of 512. Validation and training accuracy increases with epochs and error/ loss decreases which is ideal as shown in graph.

Though the model is on limited dataset and therefore can't be considered accurate but it perform very well on the given small dataset.

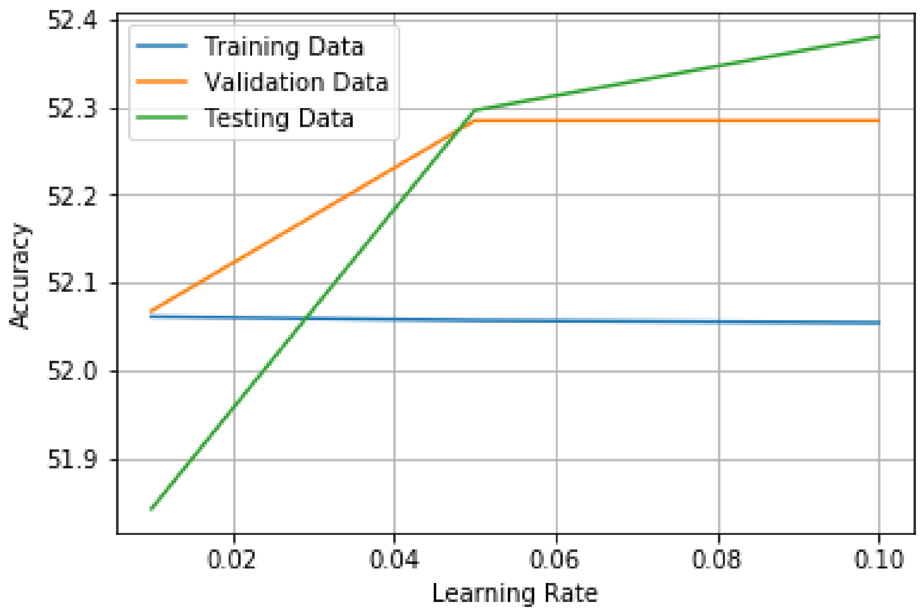
ii) Concatenated Data

1. Linear Regression

Cluster Size is chosen as 10, to capture variance of the dataset.



ERMS remain more or less constant for learning rate 0.001 – 0.1 for training , testing and validation data set.

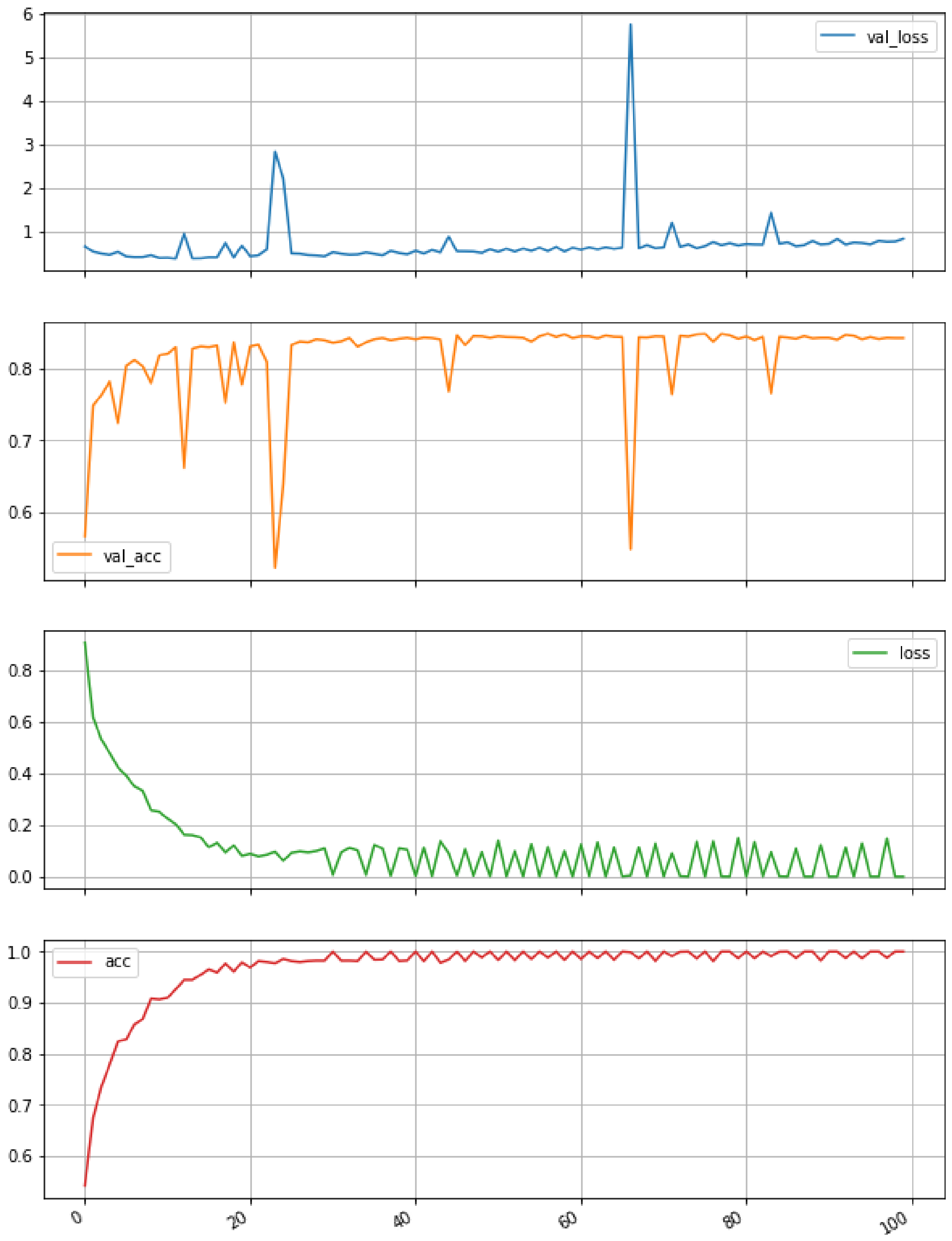


Accuracy is at peak for validation, testing and training data at **0.05 learning rate** and keeps on increasing for testing but became constant for validation and testing.

2. Logistic regression

Accuracy is calculated using the scipy.optimize library minimize function is observed as follows:
Accuracy training = 48.3%, Accuracy testing = 49.53%, Accuracy validation = 49.65%
This low accuracy could be because of the small dataset used for model building.

3. Neural Network



Testing Accuracy of 48% is observed for dense layer of 512. Validation and training accuracy increases with epochs and error/ loss decreases which is ideal as shown in graph. There is some strong noise that is also observed for some epoch but general trend is ideal.

Refrences

1. https://en.wikipedia.org
2. https://www.johnwittenauer.net/machine-learning-exercises-in-python-part-4/