Project 2- Classification

Implementation and evaluation of classification algorithms

Name: Sneha Balakrishnan Thazhathethil

UBID: 50169238

UB email id: snehabal@buffalo.edu

Objective:

The objective of this project is to recognize a 28x28 grayscale handwritten image and identify it as a digit among 0 to 9. This has to be achieved by doing the following tasks:

1) Implement logistic regression, train it on MNIST digit images and tune hyperparameters.

2) Implement single hidden layer neural network, train it on the MNIST digit images and tune hyperparameters such as the number of units in the hidden layer.

3) Use a publicly available convolutional neural network package, train it on the MNIST digit images and tune hyperparameters.

Logistic Regression:

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). In logistic regression, the dependent variable is binary or dichotomous, i.e.it only contains data coded as 1 (TRUE, success etc.) or 0 (FALSE, failure etc.).
The goal of logistic regression is to find the best fitting (yet biologically reasonable) model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. [1]

Using 1 of K coding scheme, t= $\{t_1, t_2, \ldots t_k\}$ for multiclass classification model where $a_k = w_k^T x + b_k$ can be represented as follows:

$$p(\mathcal{C}_k|\mathbf{x}) = y_k(\mathbf{x}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

Cross entropy error function for multiclass classification problem seeing the training sample x: here yk=yk(x)

$$E(\mathbf{x}) = -\sum_{k=1}^{K} t_k \ln y_k$$

Gradient of error function is given by:

$$\nabla_{\mathbf{w}_j} E(\mathbf{x}) = (y_j - t_j)\,\mathbf{x}$$

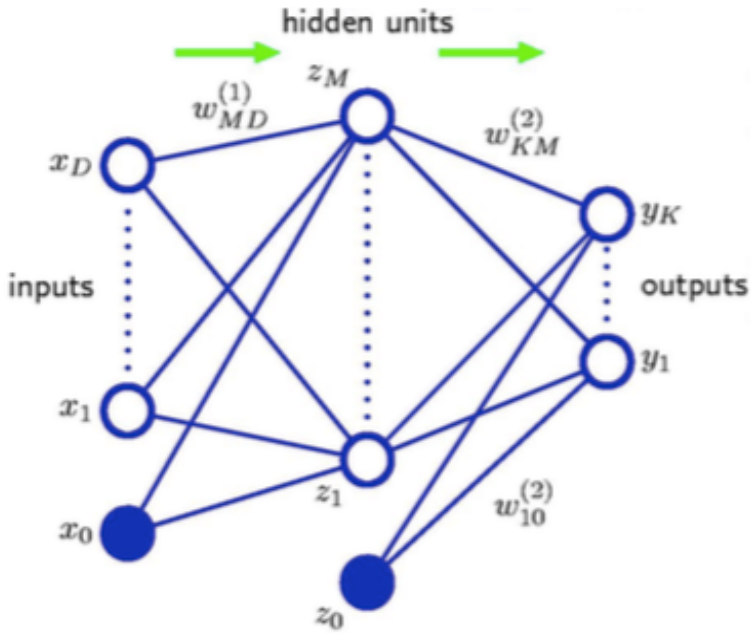
Stochastic gradient descent to find optimum of error function and find solution for w_j.

$$\mathbf{w}_j^{t+1} = \mathbf{w}_j^t - \eta \nabla_{\mathbf{w}_j} E(\mathbf{x})$$

Stochastic gradient uses first order derivatives to update.

Single Layer Neural Network:

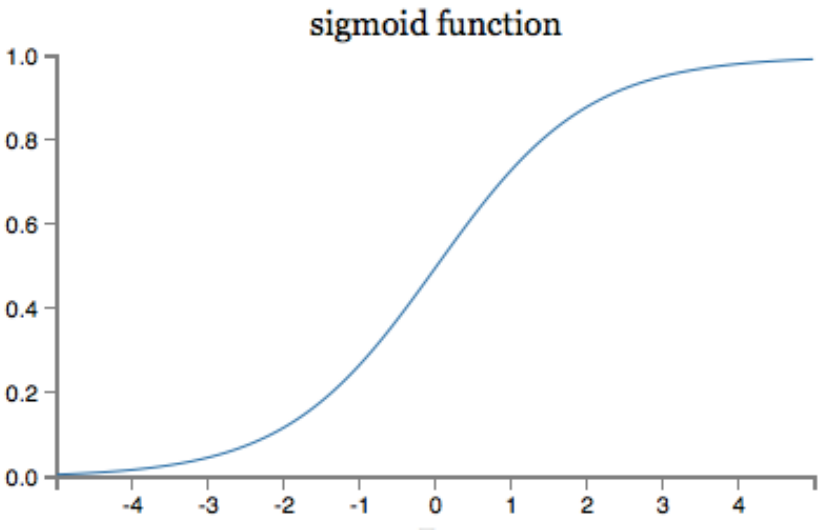
Neural network is a system of programs and data structures that approximates the operation of the human brain. A neural network usually involves a large number of processors operating in parallel, each with its own small sphere of knowledge and access to data in its local memory. Typically, a neural network is initially "trained" or fed large amounts of data and rules about data relationships. A program can then tell the network how to behave in response to an external stimulus or can initiate activity on its own. [3]



In single hidden layer neural network, input layers are represented as x_i in the model we are building and output is y_k. The feed forward propagation is given as follows:

$$z_j = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

zj is the activation of the hidden layer and h(.) is the activation function of the hidden layer. We can choose logistic sigmoid, hyperbolic tangent or rectified linear unit for activation function of which we have implemented the sigmoid function.



Cross entropy function is calculated as below:

$$E(\mathbf{x}) = -\sum_{k=1}^{K} t_k \ln y_k$$

where a_k is calculated as given below:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_k^{(2)}$$

And $y_k = y_k(x)$ is given as discussed earlier.

$$y_k = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

Backpropagation:

$$\delta_k = y_k - t_k$$

$$\delta_j = h'(z_j) \sum_{k=1}^{K} w_{kj} \delta_k$$

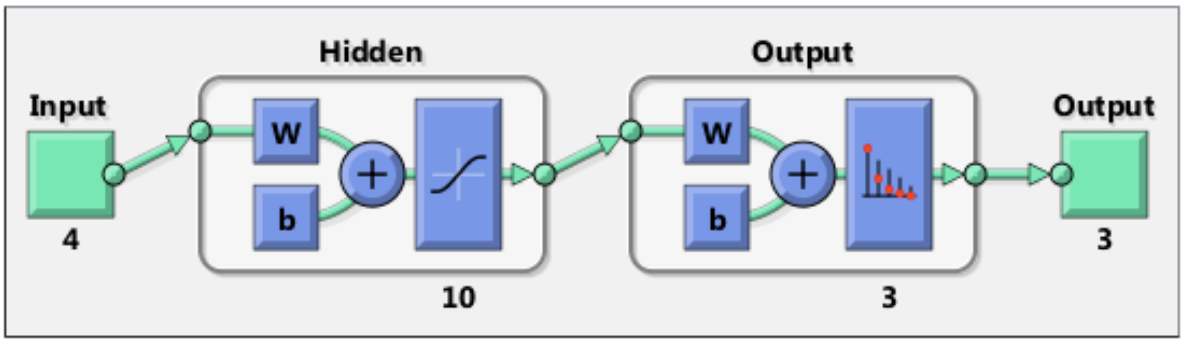
Gradient of error function is given as:

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E}{\partial w_{kj}^{(2)}} = \delta$$

Having the gradients, we will be able to use stochastic gradient descent to train the neural network. Where w are all the parameters of the neural network.

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} E(\mathbf{x})$$

By using the feature of view(net) we can see the Neural Network as below:



Convolutional neural network:

Typically, convolutional layers are interspersed with sub-sampling layers to reduce computation time and to gradually build up further *spatial* and *configural* invariance.

At a convolution layer, the previous layer's feature maps are convolved with learnable kernels and put through the activation function to form the output feature map. Each output map may combine convolutions with multiple input maps.

$$\mathbf{x}_j^\ell = f\left(\sum_{i \in M_j} \mathbf{x}_i^{\ell-1} * \mathbf{k}_{ij}^\ell + b_j^\ell\right),$$

Mj represents a selection of input maps and the convolution is of valid border handling. Each output map is given an additive bias b, however for a particular output map, the input maps will be convolved with distinct kernels. That is to say, if output map j and map k both sum over input map i, then the kernels applied to map i are different for output maps j and k.

For computing the gradients for each map j in the convolutional layer, pairing it with the corresponding map in subsampling layer:

$$\boldsymbol{\delta}_j^\ell = \beta_j^{\ell+1}\left(f'(\mathbf{u}_j^\ell) \circ \mathrm{up}(\boldsymbol{\delta}_j^{\ell+1})\right)$$

l - layer of convolution layer

up(.)- up sampling operation eg: Kronecker function as up(x) ≡ $x \otimes 1_{nxn}$

$$\frac{\partial E}{\partial \mathbf{k}_{ij}^\ell} = \sum_{u,v} (\boldsymbol{\delta}_j^\ell)_{uv} (\mathbf{p}_i^{\ell-1})_{uv}$$

p_i^{l-1} is the patch in x_i^{l-1} that was multiplied elementwise by k_{ij}^l during convolution in order to compute the element at (u,v) in the output convolution map x_j^l.

Subsampling layers:

A subsampling layer produces downsampled versions of the input maps. If there are N input maps, then there will be exactly N output maps, although the output maps will be smaller.

$$\mathbf{x}_j^\ell = f\left(\beta_j^\ell \operatorname{down}(\mathbf{x}_j^{\ell-1}) + b_j^\ell\right),$$

down(.) represents the subsampling function

If the multiplicative bias is given by β:

$$\frac{\partial E}{\partial \beta_j} = \sum_{u,v} (\boldsymbol{\delta}_j^\ell \circ \mathbf{d}_j^\ell)_{uv}.$$

Additive bias is given by b:

$$\frac{\partial E}{\partial b_j} = \sum_{u,v} (\boldsymbol{\delta}_j^\ell)_{uv}.$$

Learning combinations of Feature maps:

We sum of several convolutions of different input maps provides an output map. Consider a_{ij} as the weight given to map i while forming the map j. In this case, the output map is given as below:

$$\mathbf{x}_j^\ell = f\left(\sum_{i=1}^{N_{in}} \alpha_{ij}(\mathbf{x}_i^{\ell-1} * \mathbf{k}_i^\ell) + b_j^\ell\right)$$

Provided, the constraint below constraints are satisfied:

$$\sum_i \alpha_{ij} = 1, \quad \text{and} \quad 0 \le \alpha_{ij} \le 1.$$

and $$\alpha_{ij} = \frac{\exp(c_{ij})}{\sum_k \exp(c_{kj})}.$$

Enforcing sparseness combinations:

Adding a regularization penalty to the final error function can help include a sparseness constraint on the distribution of weights for a given map.

Error for a single pattern can be given as:

$$\tilde{E}^n = E^n + \lambda \sum_{i,j} |(\boldsymbol{\alpha})_{ij}|$$

Calculating regularization penalty:

$$\frac{\partial \Omega}{\partial c_i} = \sum_k \frac{\partial \Omega}{\partial \alpha_k} \frac{\partial \alpha_k}{\partial c_i}$$

Penalized error function can be calculated by combining the above two equations.

$$\frac{\partial \tilde{E}^n}{\partial c_i} = \frac{\partial E^n}{\partial c_i} + \frac{\partial \Omega}{\partial c_i}.$$

Dataset interpretation

The MNIST dataset that has been used to train the model in this project is a large database of handwritten digits that is commonly used for various image processing techniques and machine learning. The database consists of 60000 training images and 10000 testing images.

The MNIST dataset is already partitioned into training and test dataset. I have trained my model using this partitioned dataset itself.

Four files are available at the link provided for MNIST dataset.

train-images-idx3-ubyte.gz: training set images
train-labels-idx1-ubyte.gz: training set labels
t10k-images-idx3-ubyte.gz: test set images
t10k-labels-idx1-ubyte.gz: test set labels

Hyperparameters, Model complexity, regularization and performance evaluation

Hyperparameters are merely chosen by incrementing/decrementing(depending on the performance) and testing the misclassification rate that it provides on the training dataset.

The learning rate eta determines how “quickly” the gradient updates follow the gradient direction. If the learning rate is very small, the model will converge too slowly; if the learning rate is too large, the model will diverge.
Number of units is also a hyperparameter but in our project we have used single later neural network.
Activation sparcity is implemented in convolutional neural network.

Model complexity can be controlled by adding a regularization term. Network complexity will add to over fitting which is to be avoided. The simplest regularizer is weight decay. This is done by introducing regularization coefficient lambda.
Regularization can be done by determining the number of hidden units. Number of hidden units is a free parameter. It can be adjusted to get best predictive performance.
Epochs:

As seen in the misclassification data provided by the autograder for the different models which have been build. The misclassification rate for neural networks is comparatively lesser than logistic regression approach. Though both the methods have their own pros and cons, to recognize digits, neural network approach is better for performance related challenges as it can be tuned to suit the requirements by just changing the number of hidden layers.

References:

http://www.inf.ed.ac.uk/teaching/courses/inf2b/learnnotes/inf2b-learn-note11-2up.pdf

[1]https://www.medcalc.org/manual/logistic_regression.php

http://colinraffel.com/wiki/neural_network_hyperparameters

[3]http://searchnetworking.techtarget.com/definition/neural-network

http://cogprints.org/5869/1/cnn_tutorial.pdf

http://yann.lecun.com/exdb/publis/pdf/lecun-90c.pdf

https://github.com/rasmusbergpalm/DeepLearnToolbox/blob/master/REFS.md

http://www.academia.edu/336751/Complex_Neural_Networks_-_A_Useful_Model_of_Human_Learning

Machine learning slides posted on UB learns.

http://www.mathworks.com/help/nnet/ref/view.html

http://neuralnetworksanddeeplearning.com/chap3.html

http://www.worldscientific.com/worldscibooks/10.1142/5345