CS 419M Introduction to Machine Learning **Spring 2021-22**

Lecture 16: Introduction to Deep Learning

Lecturer: Dr. Ashish Tendulkar (guest) *Scribe: Group 1*

16.1 Introduction to Deep Leaning

Deep Learning also called as Artificial Neural Networks, Deep Networks, Multi-layer Preceptrons (MLP) is a machine learning model like Linear Regression, Logistic Regression, Decision Trees, SVMs .

Deep Learning Models are ML models only.

- **What is Machine Learning? What is the difference between Programming and ML?**
  - Programming: When process is simple with given data and specified rules to obtain output. example, adding of two numbers. Writing a code to detect human faces in an image is difficult by this approach cause of brittle rules.

Figure 16.1: Programming Flow Chart

  - Machine Learning: When rules are not well defined, are brittle and its difficult to write the program in the traditional way. We have ML to the rescue.

Figure 16.2: ML Flow Chart

    * We provide training data for the task at hand - Detecting human faces from the images.

* Training data is in the form of input image and output binary. like (image1,label1), (image2,label2) — (imagen,labeln)
* Let's say image is of pixel 32*32 than input (x) is a vector of length 1024
* Output (y) is a binary
* Training data gives us the weights of the models. (where models are specified)
* After training we get inference which takes the weights and input data to give output similar to the traditional program.

16.2 Components of ML

Neural Network (NN) is just another machine learning concept. They also have 5 components:

- Training data (consists of features and labels) :
  - Regression: training data is of the form $\left(\vec{X}_i, y_i\right)$, where $y_i \in \mathbb{R}$. If only one label $\rightarrow$ single output, otherwise $\rightarrow$ multi-output.
  - Classification: training data is of the form $\left(\vec{X}_i, y_i\right)$, where $y_i \in$ a discrete set. Within classification we have binary(single label) and multiclass classification(multi-label).
- Model:
  - Linear Regression: $y = w^T x$
  - Logistic Regression: $y = sigmoid(w^T x)$
  - Decision Trees: Tree (By traversing the tree we can reach leaf node of the tree and based on the label in leaf node we can assign the label)
- Loss functions:
  - Regression: MSE (Mean Squared Error) loss :-

$$\begin{aligned} J(w) &= \frac{1}{2}\sum_{i=1}^{n}(prediction - actual\,label)^2 \\ &= \frac{1}{2}\sum_{i=1}^{n}(w^T x - y)^2 \\ &= \frac{1}{2}(Xw - y)^T(Xw - y) \end{aligned}$$

  - Classification: Cross Entropy loss:- $J(w) = \sum_{i=1}^{n}(actual\,label)\,log(predicted\,value)$
- Optimization procedure
  - Update rules

- Gradient descent, batch gradient descent, Stochastic Gradient Descent (SGD) and Greedy method for trees

- Model selection and evaluation
  - Hyper-parameter tuning (HPT): For HPT, we can use grid search, randomized search.
    - Regularized LSE:- $J(w) = J(w) + \lambda * Penalty$
      Vary value of $\lambda = (1e^{-4}, 1e^{-3}, 1e^{-2}, 1e^{-1}, 0, 1)$
      $(\lambda, MSE) \rightarrow Select\, \lambda$ that gives best MSE(calculation done on validation set)
    - The only difference between both the approaches is in grid search we define the combinations and do training of the model whereas in Randomized Search the model selects the combinations randomly.
  - Metrics:
    - Regression: MSE, MAE
    - Classification: Accuracy,Precision,Recall,F1 score

16.3 TensorFlow - A Neural Network Playground

16.3.1 Examples

We can visualise machine learning "in action" using this website. Here are a few examples covered in class :

1. We have two classes, blue and orange, we have chosen the Gaussian dataset, and kept the ratio of training to test data as 70%. We can also configure the noise and batch size (in case of mini batch gradient descent). We keep the noise at 0 and make the batch size 16. We have two features - x_1 and x_2 and the problem type is classification. We discretize the output and use logistic regression model i.e. set activation to sigmoid. When we trai our model, we see that after every epoch, there is a decrease in training and test lost. As we increase the learning rate, the number of iterations taken to reach a certain loss value decreases.

2. We now use the circular dataset. Since, in real world we will be unable to visualise the data, we try forming conclusions based on the training and test loss.We run the same model as in the first example and observe that the losses saturate. The training and test losses are high and hence, we infer that the model is **underfitting**. To remedy this, we increase the complexity by adding more parameters.

   (a) We add higher order terms - x_1^2, x_2^2 and x_1x_2. We are thus able to 'learn' the circular distribution boundary. To reduce over-representation, we use regularisation.

   (b) We use **neural networks**. We will not need higher order terms in this case. We add two hidden layers. This is the architecture of the model and is called FFNN. We change activation to ReLu and there is linear combination followed by non-linear activation. To ensure that model is not underfitting, we add another neuron (we could've also added another layer). Now, we have 20 parameters, thus increasing the computation.

16.3.2 Key Learnings

- To solve the problem of underfitting, we need to increase complexity by adding paramters. Getting more data will not be of any use.
- Neurons are the basic units of computation of a hidden layer. These layers are called hidden since the true values of their nodes and unknown.
- Neural networks are prone to overfitting because of their excess capacity.

16.4 Components of Neural Networks

Neural Network (NN) is just another machine learning concept. They also have 5 components:

- Training data:
  - Regression: training data is of the form $\left(\vec{X}_i, y_i\right)$, where $y_i \in \mathbb{R}$. Linear activation function is used in the output layer for regression problems
  - Classification: training data is of the form $\left(\vec{X}_i, y_i\right)$, where $y_i \in$ a discrete set. Sigmoid activation function is typically used in the output layer for classification problems
- Model: the architecture of a NN is defined by the the number of hidden layers, and the number of neurons/units, and the way they are connected to each other. Each unit is also known as a perceptron. Based on the architecture, we can have various kinds of NNs:
  - Feed-forward NNs (FFNNs), also known as multi-layer perceptrons. FFNNs are what we encountered in the lecture
  - Convolutional NNs (CNNs): these are commonly used in image processing and image understanding problems. 1-D analogue of CNNs can also be used for time-series problems
  - Recurrent NNs (RNNs): these are often used in sequence modelling/time series modelling. Examples of such NNs include LSTMs, GRU, transformer models, attention models

  Combinations of different types of NNs can also be used. For instance, in video processing, here is what is typically done:

  $$Video \rightarrow CNNs \rightarrow RNNs \rightarrow FFNNs \rightarrow labels$$

  In a typical ML formulation, suitable feature vectors have to be constructed by a user. The model looks like:

  $$Examples \rightarrow features \rightarrow models \rightarrow labels$$

  A power of NNs is that feature representation is also possible. That is, suitable features do not have to be explicitly provided. The model can implicitly find a suitable representation on its own:

  $$Examples \rightarrow model \rightarrow labels$$

- Loss functions: similar to ML
  - Regression: MSE (Mean Squared Error) loss
  - Classification: BCE (Binary Cross Entropy) loss, CCE (Categorical Cross Entropy) loss. The latter is used for multi-class classification problems
- Optimization procedure
  - Gradient descent, batch gradient descent, Stochastic Gradient Descent (SGD)
  - Specialized techniques for NNs: Adam, adaptive, momentum-based techniques
- Model selection and evaluation
  - Hyper-parameter tuning (HPT): number of layers and number of neurons are also like hyper-parameters. For HPT, we can use grid search, randomized search. HPT can also be performed on Google cloud (known as Vizier)
  - The evaluation metrics are similar to that in conventional ML

A classic deep learning problem is digit identification using the MNIST dataset. The training data contains a set of images with each image representing a hand-drawn digit, and the corresponding label, which is just the digit drawn. The problem is to identify the digit which is drawn from a given hand-drawn image. FFNN based solution to this problem was demonstrated (on Google colab) in the lecture.

16.5 Group Details and Individual Contribution

Name	Roll Number	Contribution
Laxita Karnawat	200110063	Section 16.1 + Submission on Github
Ankit Yadav	190100019	Section 16.2
Vedika Gupta	20B090013	Section 16.3 + Compilation on LaTeX
Himansh Rathore	180260015	Section 16.4