Data Mining

STAT 557

Fall 2018

Survey Report

Machine Learning and Deep Learning

Submitted by: Nikhil Sunil Nandoskar

nxn59@psu.edu

994844501

Professor: Dr. Jia Li

Introduction:

Machine Learning is changing our day to day life. Technology is improving at a faster rate due to machine learning. In machine learning we build algorithms and provide them with data to train them for a particular task. In machine learning we have three types of learning viz Supervised Machine Learning in which we always have a ground truth/ labels and improve our model's prediction by comparing them with these ground truths. Other type of learning is Unsupervised Machine Learning where-in we let the model learn on its own without supervision. They are classified in two groups: 1) Clustering 2) Association. After this comes Semi-Supervised Machine Learning. In such type of learning we have a huge amount of data and some part of it is labeled. Feature Engineering is a crucial part of Machine learning. It is always beneficial to have a clean data for good results however, it is very expensive and time consuming to label a huge amount of data. In such situation semi-supervised machine learning plays a vital role. Machine Learning is a vast field and there are various algorithms available today. Researches are continuously trying to improve existing algorithms and trying to solve complex word problems. Due to advancement is computational power, hardware we now have more flexibility to build more complex models. This has led to growth of Neural Networks which is a part of machine learning tackling real time scenario issues. The Neural Networks are good at handling images and video data and perform tasks as face detecting, image classification, autonomous cars, medical industry. The following sections contains information regarding Supervised Machine Learning and Neural Networks.

Supervised Machine Learning:

In supervised machine learning input data as features, also called as independent variables and we have a dependent variable, the output. Here our model tries to learn the mapping function of our input to output. The feature engineering steps involved here are splitting the data into first input feature and output variables also classed as ground truth/ labels. After then we divide our entire dataset into training dataset and testing dataset. Machine learning has an analogy with the working of human neurons. In machine learning we feed the data to the input layers. These input layers are associated with a random weights and biases and connected to so called neurons which form the hidden units of our model. The neurons in these hidden layers compute the summation of the products of the input data and weights and biases and generates outputs. These outputs are then given to a hidden layer which decides whether a neuron will fire or not depending on the threshold set. Various activation functions used in machine learning are as follows:

Sigmoid activation function: $\frac{1}{(1+ e^{-z})}$

where: z = weights*inputs features + bias

Tanh activation function: $\frac{e^{+z} - e^{-z}}{e^{+z} + e^{-z}}$

ReLu activation function: max (0, z)

In practice ReLu is used extensively for hidden layers as it imposes non-linearity and tackles vanishing gradient problem. In a machine learning algorithm, we have few hidden layers. The output layer gives us the prediction. In supervised machine learning the loss is then calculated. The various loss/cost function used in machine learning are as follows:

Mean Squared Error: $\sum_{i=1}^{n} \frac{(predicted\ output - ground\ truth)\text{^}2}{n}$

where: n is the number of samples in training

Cross Entropy Error: It is used in classification tasks. It measures the performance of a classification model output is a probability between 0 and 1.

For binary classification the loss function is:

Loss = -(y_true log(y_predicted) + (1-y_true) log(1-y_predicted))

In case of multiclass classification, the loss function is given as:

Loss = $-\sum_{i=1}^{n} y\_true_{o,i} * \log(y\_predicted_{o,i})$

These errors are then backpropagated to adjust the weights and biases in order to improve accuracy. This procedure is called as Backpropagation. In machine learning we always try to minimize this loss. In machine learning we use various optimizing techniques to minimize these errors. One such is Stochastic

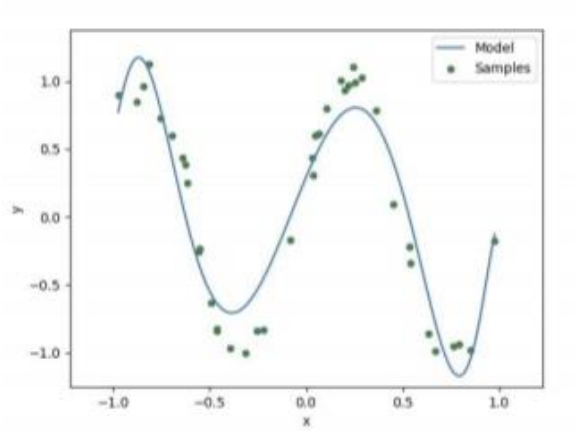
Gradient Descent (SGD). In SGD we take one sample at a time and compute the above steps, but this is time consuming so we go for Mini Batches where our algorithm randomly pics mini batches of examples and computes the error on the entire mini batch. Here we consider a convex problem and assuming initially we are at a certain point on the convex shape, our aim is to reach the global minima in order to obtain stability and best accuracy. The speed with which we reach the minima is guided by the hyperparameter learning rate α. We even regularize the learning by adding an extra hyperparameter λ (lambda) that penalizes the loss function from overfitting. In machine learning we always have a problem of bias v/s variance trade off. The following table will explain the concept of bias v/s variance

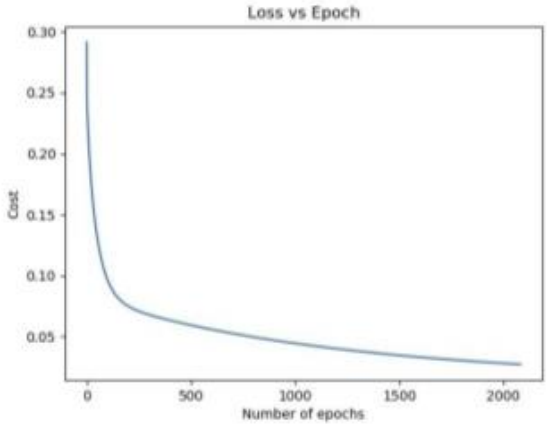
Observations	**Error (%)**	**Result**
Training Set	1	Low Bias High Variance
Testing Set	16	
Training Set	15	High Bias Low Variance
Testing Set	16	
Training Set	16	High Bias High Variance
Testing Set	30	
Training Set	0.5	**Low Bias** **Low Variance**
Testing Set	1	

Table 1: High Bias and Variance Problem

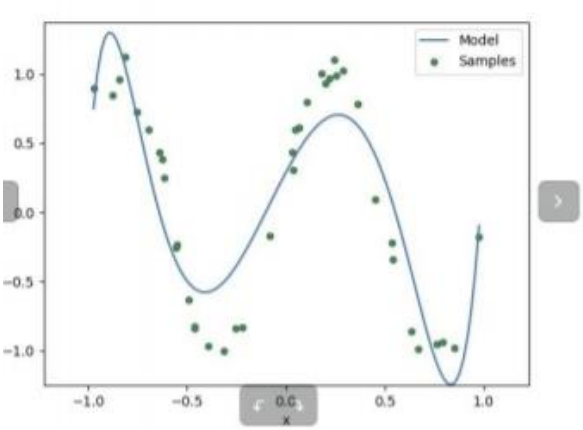
From Table 1 we observe that we always want a low bias and low variance for our model. If we get a very good accuracy on our training set but very poor accuracy on testing set then our model is *'overfitting'*. Solution to this is using 'Regularization'. There are various types of regularization techniques used in machine learning like L1 regularization where in it penalizes the loss function by making weights zero i.e., we get a sparse matrix. So, our model complexity decreases. In L2 regularization we compute squared errors of our loss function. L2 penalizes higher difference of weights with higher penalty and lower difference with lower penalty. Another regularization technique we use is Elastic net which take advantages of both L1 and L2 regularization. There are various other techniques to improve performance of our model viz Batch normalization (to avoiding weights and biases from exploding), Principal component analysis (for dimensionality reduction), k-fold cross validation (training on k-1 slices of data and testing on k^{th} slice and repeating this process for k iterations).

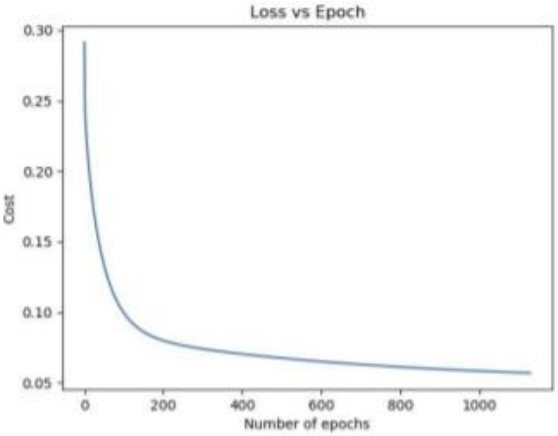
Supervised Machine Learning is divided into following two categories:
Regression: When we have a continuous output, we use regression.



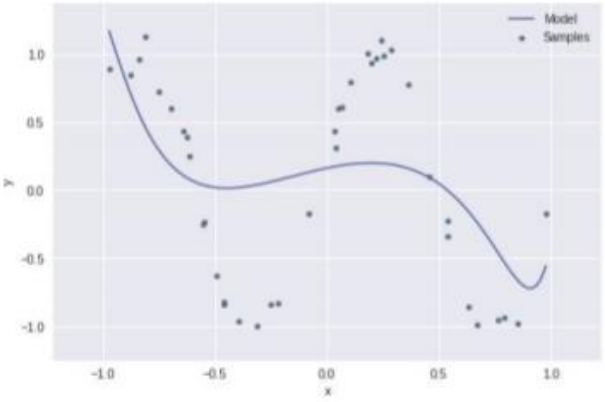


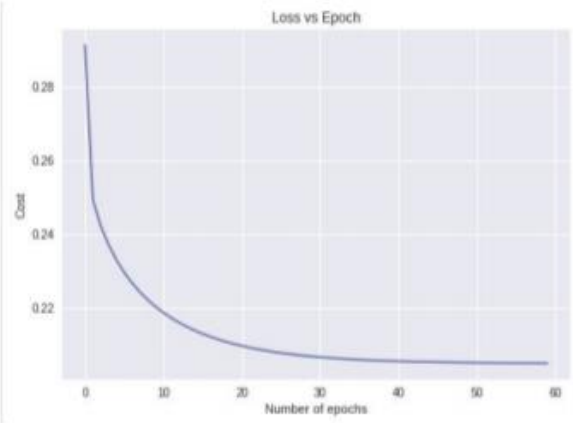
When λ = 0.001





When λ = 0.01



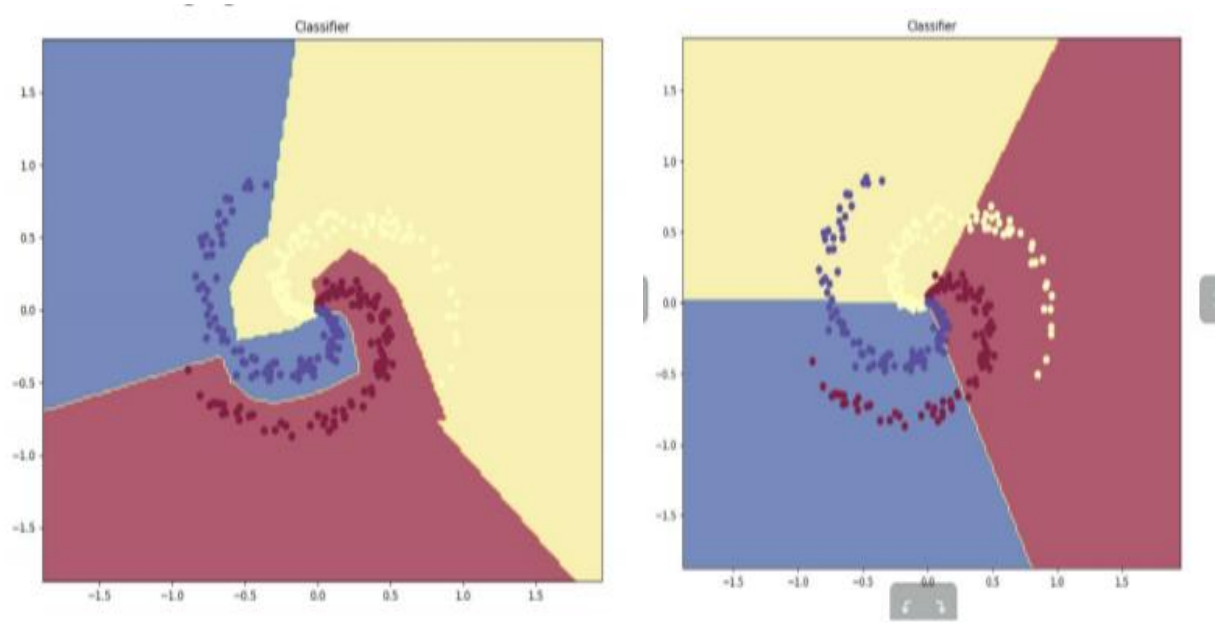


When λ = 1

Figure 1: The learning rate is 1.5 and Degree of polynomial is 15

The above graphs convey the importance of Regularization in Machine Learning. It used to avoid overfitting our model to the given data or else it will perform poorly on testing data.

Classification: Whenever the output is categorical or discrete, we use classification.



(a) (b)

Figure 2: Spiral data

The above plot is of spiral data in which we have three classes. The above plot conveys the importance of hidden layers in Machine learning. We can clearly see this as for (b) the decision boundaries are linear as there is no hidden layer in it. The hyperparameters chosen for (a) were α = 1, λ = 0.0001

Deep Learning:

Deep learning flourished in recent years due to availability of huge chunk of data, improved algorithms and strong computational power. Convolutional Neural Network (CNN) is a special kind of deep learning network which is responsible for recognizing visual patters directly from pixel images. The CNN are developed to extract relevant features on their own instead of using a special engineer to do this task. The depth i.e. the number of layers in CNN is an important factor to consider. Simpler CNN cannot be used to solve complex problems. This led to invention of Deep Convolutional Neural Networks (DCNN). DCNN are used in Image-recognition, classification task. There are various DCNN available in today's date. They developed in the mid 2000's beginning from AlexNet, GoogleNet/Inception, VGGNet, ResNet. Researchers always try to either modify existing architecture or come up with an altogether different new architecture to solve real world problems using Deep Learning. Nevertheless, the base architecture remains the same when we compare DCNN with simple CNN. In CNN we have convolutional filters which are applied on input images, immediately after this we add a ReLU non-linear activation function to introduce non-linearity. After this we apply pooling to extract more relevant pixels. We even apply batch normalization just to make sure that our weights and bias don't explode while training. The above steps are repeated several times in order to create a Deep Neural Network. After the final pooling layer of our DNN we flatten out the outputs and feed them as inputs to a Fully Connected Neural Network (FC). The loss function of our model is defined in the FC layer. The following section contains details of various layers of DCNN.

Convolutional Filters:

Convolution filters sometimes referred as kernels are convoluted over an input image of size height x width x channels to extract important pixels and form a feature map which is used during learning phase. The convolution operation is given by the following formula

Feature Map = $f($ N * I)

where N = Convolution filter related a feature map

I = Input image

* = 2D convolutional operator

In CNN at every step the output size of image decreases if padding is not provided. Suppose we have an image of size 6 x 6 and filter/kernel size of 3 x 3, then the convolution operation will output an image of size 4 x 4. This is given by a formula

output image = image size – kernel size + 1

In DCNN if we keep doing this, we will soon land up having no pixel left for computation for next deep layers. Thus, we need to zero pad our input image. Other reason for padding input image is when we stride over our input image the pixels present on the edges are only visited once whereas the pixels present in center part of the image are seen frequently. In Tensorflow for conv2D library function we set the padding parameter to 'Same'. It pads the input in such a way that we get the output image as same size of input image. The size of output image is given by following formula

output image = (image size – 2*padding + kernel size)/stride + 1

ReLU Activation Function:

In CNN after apply convolution on the input image we apply ReLU function in order to introduce non-linearity in the model. Experimental results convey that ReLU performs better than sigmoid and tanh activation functions. ReLU speeds up learning process and tackles vanishing gradient problem (where weights don't really change for lower layers during backpropagation). The ReLU function is as follows

f(x) = max(0, x)

Pooling:
After applying ReLU we make our image go through various pooling layers. There are two types of pooling techniques used viz Average pooling and Maximum pooling. In this layer we stride over our convoluted image picking up relevant pixels and discarding other pixels. In practice Maximum pooling also termed as max pooling is used where we take only the maximum pixel in a given stride. Pooling helps in reducing computational cost as irrelevant data is dropped.

Batch Normalization:
In general, we normalize our input features so that our neural network train properly. For images we normalize the RGB values of our image before training. But the weights of hidden layers can explode during training process. This can be avoided using batch normalization. It does this by subtracting the batch mean and dividing by the batch standard deviation for the output of previous activation layer

Fully Connected Layers:
The above-mentioned steps are repeated several times, thus stacking layers on each other, abstracting more information and features of inputs. After taking the batch normalization of the final convolutional layer we flatten out the output and feed it to a fully connected layer. The number of layers of fully connected layers depends on the inventor. The total number of neurons and output classes of the final layer of this fully connected layer depends on a particular problem statement. We again use a ReLU activation functions except for the last layer where we use Softmax activation function as we want to have a probabilistic distribution to confirm as to which class our model is predicting the input belongs to. The Softmax activation function is given by the following formula

$$\mathbf{p}_k = \frac{e^{\mathbf{f}_k}}{\sum_j e^{\mathbf{f}_j}},$$

The summation in the above formula is along axis = 1/ along columns of the matrix
The Loss function is given by the following formula

$$\mathcal{J} = -\frac{1}{N}\sum_i \Big(\log(\mathbf{p}_{y_i})\Big) + \frac{\lambda}{2}\sum_d\sum_k (W_{d,k})^2$$

The λ is a hyperparameter for regularization which is used to avoid overfitting

Architecture of VGG-16

We have various options to choose between VGG-16, VGG-19, Inception, ResNet. However, when we want less complex model and the dataset is less as compared to the original dataset (Imagenet) used for training the VGGNets by the inventors, we choose VGG-16. Tensorflow, Keras framework can be used to build the model from scratch. The architecture of VGG-16 is as follows:

Input Image (2248224*3)
Conv3-64 Conv3-64
Max-pooling
Conv3-128 Conv3-128
Max-pooling
Conv3-256 Conv3-256 Conv3-256
Max-pooling
Conv3-512 Conv3-512 Conv3-512
Max-pooling
Conv3-512 Conv3-512 Conv3-512
Max-pooling
FC-4096
FC-4096
FC-353
Softmax

We use images to be of size 224*224*3 in order to maintain the kernel size of 3*3 and stride 1 for the convolutional layers. Interesting point to note here is the kernel size is fixed throughout. The reason for it is that when we stack three 3*3 convolutional layers with a stride of 1 we have same effective receptive field as 7*7 convolutional layer. Max-pooling is performed over 2*2 pixel window with a stride of 2. More interestingly in our model we are using batch normalization for the reasons stated earlier. As batch normalization concept was not discovered when VGG was originally invented. Also, dropout was

not used then. We can use a dropout of 40% to avoid overfitting. In dropout our model randomly mutes few neurons outputs. For learning, we can use Adam optimizer as it takes advantages of both AdaGrad and RMSProp. Unlike, Stochastic Gradient Descent (SGD), Adam changes its learning rate as the learning unfolds. As the original model was train on 1 Million images it took 2 weeks to train and test on 4 GPUS. Such powerfull GPUS are not available to the students, we can try Fine tuning or Transfer learning and train the VGG-model on our research dataset. In Fine tuning we free initial layers and try to tune only the last layers as required. In transfer learning we change the output layer only i.e. the Softmax layer as per the number of classes we have for our problem statement and train the model again. In tensorflow while building a model from scratch we use *tf.nn.softmax_cross_entropy_with_logits_v2* function which performs softmax along with cross entropy at the same place. However, in pre-trained model we already have softmax applied so we just need a function only performs cross entropy loss function. Such a function is available in Tensorflow called *tf.losses.softmax_cross_entropy.* We are using this in our transfer learning model implementation.

Conclusion:

My area of expertise is Deep Learning. I wanted to make my Machine Learning concepts strong and learn the math involved in it. The benefit I got taking this course was I learned new Machine Learning techniques like K-NN, Decision-Tress, Bagging, Boosting, XG-Bosst and applied them in my project. The project also improved my skills in using pandas, sklearn, python coding.

References:

[1] Class notes

[2] Dr. Andrew NG Machine Learning Course

[3] Very Deep Convolutional Networks for Large-Scale Image Recognition

[4] Goodfellow, I., Bengio, Y., And Courville, A. Deep Learning. MIT Press, 2016