’’’

Machine Learning Design

Author: Tyler Travis, Supervisor: Dr. Huajie Zhang
Undergraduate Program of Software Engineering
University of New Brunswick, Fredericton, NB, Canada
Email: *ttravis@unb.ca

***Abstract*—In compliance with requirements for the University of New Brunswick's SWE4913 curriculum. The purpose of this independent research project is to build knowledge on basic concepts of machine learning models.**

***Index Terms*—Machine Learning, Machine Learning Independent Study, Machine Learning Architectures, Housing Prices, Autonomous Vehicles**

I. Introduction

After looking at a single-layer perceptron neural network in my secondary education, a simple quadratic polynomial fitter made myself question the capabilities of neural networks in computing. This opportunity to expand knowledge on such a paramount part of the technological revolution is essential to my growth as a software engineer.

This paper is cohesive with a public GitHub repository with any relevant code/data to the research.

The goal of this paper is to have reference to the key elements of Machine Learning, as a guided description of my discoveries.

Topics will be drawn from academia, novels, and public GitHub repositories. Literature on Machine Learning will be useful to grasp concepts of the structural architecture and procedure of an algorithm. Albeit understanding is the most important factor, I also plan on looking at examples of machine learning code with the goal deciphering what is going on under the hood.

II. Notes of The Hundred Page Machine Learning Book

This book was chosen as it is a very popular read, and is described as a good introduction to machine learning. Its concise nature allows a general understanding, and was selected to be my first read in my deeper endeavour of ML.

Machine learning algorithms are often described as "supervised", or "unsupervised". There are also "semi-supervised" and "reinforcement" machine learning algorithms.

- **Supervised Models** uses labeled sets of data to train an algorithm, and by iterating can predict more accurately. They often take feature vectors in, and output information accordingly. It is the most commonly used learning method. The data is supervised to (input, output).
- **Unsupervised Models** are in charge of identifying patterns of unlabeled data. There is also a feature vector given into this learning alternative.
- **Semi-Supervised Models** is a mix of labelled and unlabelled data. The end goal is to build a strong algorithm.
- **Reinforcement Learning** is interpreted as a state, that can execute actions at every state. Depending on actions, a reward system is engaged accordingly. This helps a computer algorithm decipher policy. This is measured through excepted average reward.

Classification and Regression are often mixed terms. In our terms, **classification** entails automatically assigning labels to unlabeled data, where the then-labeled data can help develop a model. For example spam detection. For **regression** is the pursuit of predicting a label based off an unlabeled example. For example, judging a house price, based off location, number of bedrooms, area etc.

Machine Learning algorithms can be manually developed, however the industry standard is known to leverage libraries to source their work.

A **Neural Network** is a nested function,

$$y = f_{NN}(x) = f_3(f_2(f_1(x))) \tag{1}$$

hence the common statement of "layers" within a neural network. The vector functions follow the form,

$$f_l(z) = g_l(W_l z + b_l) \tag{2}$$

where l is the the layer index (spans from 1 to any number of layers), and g_l is the activation function. The parameters W_l is a matrix, b_l is a vector.

III. Case Study: House Prices

The first study that will be looked at through my pursuit of ML is its application to predicting housing prices. John Ade Ojo's article: "Predicting House Prices with Machine Learning" is a very organized, thorough explanation of how to identify patterns in the housing market, and predict a price dependant on it.

The pursuit of this machine learning implementation is to use a publicly available data-set of housing prices along with different aspects of the house (which will be used as our feature vector). The available data set will be used to train the model, and for that reason can be described as a supervised model.

Ojo decided to build his model in Python, taking advantage of its various assortments of libraries that are known to be useful in the language. His tools include,

- **Pandas** to structure his data properly. If data-science has taught me anything, data-wrangling is 75 percentage of the work. Pandas is very useful for data-wrangling, in the various ways it can be cleaned (NaNs, incomplete information disregarded).

- **Scikit-Learn** is one of the leaders in machine learning classification, regression, and clustering algorithms. Has close ties to Pandas, Numpy, and Scipy.

- **Numpy** is Python's leading tool for any sophisticated mathematical endeavours. It has its own array data structure that can speed up rigorous lengthy training, and is worth the difference. If that is not enough, it has basically all mathematical capabilities one could ask for, including matrix algebra.

- **Seaborn** is probably the least known tool out of those that were used. This is similar to MatPlotLib, and is used for data visualization.

Something interesting about this project is its very inconsistent flow. The order may overlap and there are occasions where going backwards is required, however the order typically is:

- **Data Ingestion**, the process of extracting the data that will be used from some source for further steps.

- **Data Cleaning**, also is commonly called data wrangling, and is dependant on data ingestion. Data wrangling can mean merging two different data sources, clearing gaps in data, or removing duplicate entries.

- **Exploratory Data Analysis** is when things start to get interesting. This is mixed with the tool of data visualization, and is often used as an indicator of the quality of the data you are using.

  The **Correlation Matrix** is matrix with the width equal to the height, of which each element is one feature. By this format, a output is shown of the Pearson's calculation for element $X_{i,j}$ (also known as a correlation calculation factor). This shows how strong of a correlation there is of two variables where X is the correlation factor that is calculated on the map between two features: i, j. There is a consistent diagonal perfect correlation which makes sense as some specific features correlation with itself should be perfect (1 in a numerical case).

- **Feature Engineering** can be a bit messy, and sometimes conjoint into the exploratory data analysis section. Ojo clearly understands good ML practices, since he used a natural logarithm to enclose outliers in pursuit of a normal distribution. Outliers tend to be bad when building models, and are not helpful for loss-functions.

- **Machine Learning** is an iterative process, especially for inexperienced users in the pursuit of machine learning. When Ojo implemented his machine learning into this project, he explains different model mechanics. For most beginner projects using smaller data sets, random forests and gradient boosting are pretty solid models.

  A major part of a successful ML algorithm is the training process, where model hyperparamaters are tweaked in an iterative effort.

  There is a bias variance trade-off. If the data is simpler, often the training data is unfit. Model variance is with more complexity and higher variance, which leads to poor predictive capacity of unseen untrained data.

IV. Autonomous Vehicles

Autonomous vehicles are a growing interest for many engineering followers. Many major car companies are looking into making this a feasible future for the average car-driver. The thought of a car being able to work on its own might seem like a surreal endeavour, but machine learning is making it a reality.

There is a lot of criticism with the thought of self-driving cars. To say self-driving cars are a necessity in society in 2022 might be a stretch, many deem it as a benefit that should be looked into however. Others put emphasis on the reduced risk of road traffic deaths, which can limit the mistakes humans would make at the wheel.

The technology behind autonomous driving is referred to as ADAS or advanced driver-assistance systems. The perception of a vehicle is similar to the perception of a text-recognition software, and is primarily different in the form of magnitude and complexity. As text-recognition software identifies and detects letters as specific objects, many sensors and cameras on cars allow a machine to recognize objects, and classify them. Some easy objects to conceptualize for ADAS systems are other vehicles, people, lanes, trees etc.

Accidents can still occur when training a ADAS model, and this is when information is classified wrong (ex: a missed car that has a color very similar to the road). This is where training the ADAS model is important. Millions of hours of supervised data trains most ADAS models, that will help the algorithm and effectively teach it to do something similar. Driving is such a complex task with millions of parameters, so for an effective algorithm, it is important to have enormous amounts of data.

There are benefits to different types of sensors. Lidar radars are more active then the traditional camera sensors. Lidar sensors are excellent at measurements (ex: distance between cars, stop sign etc). Traditional cameras are more vulnerable to weather and misinterpretation, but can also identify more than just some abstract distance.

A common conversation with self-driving vehicles are why Machine Learning should be used compared to vision systems. ADAS replicates human driving, by identifying assortments of rules and evolving through iterations. A car produces varying situations (varying input vector), but at the end the car should make the correct decision with minor differences, and this adds value to machine learning which is notorious for finding patterns.

V. My First Project

I decided that after building a stronger knowledge in ML, that I wanted to take part in a small project. I've heard really strong opinions about TensorFlow's open source library. They have thorough documentation, which helped me build my first ever model.

I understand the work I am learning from is an expansion of what they have already created and understand this work is created under the MIT Licence, and plan on abiding to all that entails.

The plan during this project is to classify articles of clothing based off a given image. 70,000 articles of clothing were used, each picture was 28 x 28 pixels (however converted later to 1 x 784 pixels).

Firstly, we will import the libraries we require. Tensorflow will be required for building the ML Model, Numpy for array manipulation, and Matplotlib will be used for graphing purposes.

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

Listing 1. Importing External Libraries

The next step is data ingestion. We plan on using a built in TensorFlow clothing dataset, to simplify the process. In future projects, I plan on implementing my own data-set.

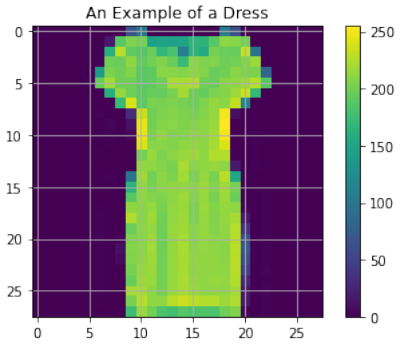
```
fashion_mnist = tf.keras.datasets.
    fashion_mnist

(train_images, train_labels), (
    test_images, test_labels) =
    fashion_mnist.load_data()
```

Listing 2. Data Ingestion MNIST Dataset

There are 10 output neurons, describing how confident the algorithm is in each specified article of clothing. From index[0] to index[9] of output neurons, they coincide with: t-shirt, trouser, pullover, dress, coat, sandals, shirt, sneaker, bag, ankle boot.

It is also important to check out our data, here is an example of the 20th index of our training images with a labeled output. This output describes a dress, and for us humans it is quite trivial. Our plan is to teach a computer in a similar way.



However, the computer prefers reading in grey-scale, so I wrote a function to do that to both our training images, and testing images.

```
def preProcessGrayScale(data):
  return data / 255.0
```

Listing 3. Grey-Scale Function

It was advised that we verify the labelled data and see its accuracy before training the model. Although hard to see, the labeled data is correct by labeling this data: ankle boots, t-shirt, t-shirt, dress.

Now for the juicy work, building a neural network model. We will use TensorFlows' Keras library, to build a sequential machine learning model. A sequential model is used when each layer has one input tensor and one output tensor. In any Machine Learning application where there is layer sharing, I will refrain from using a sequential model,

```
model = tf.keras.Sequential([
    # Switches 28 x 28 to 1 x 724
    tf.keras.layers.Flatten(input_shape
        =(28, 28)),

    # RELU activation function
    tf.keras.layers.Dense(128, activation
        ='relu'),

    # Densly connected neural networks
    tf.keras.layers.Dense(10)
])
```

Listing 4. Building Machine Learning Model

After flattening a layer into 1 x 784, and then having two dense layers of which the middle layer uses the RELU activation function, and the output layer proceeds to give confidence in all outputs, it is time to train the model.

By training the model, many steps are nested into this. 1. We are feeding training data to the model, 2. we are asking the algorithm to make a prediction, 3. we are checking the answer.

We used the Adam Optimizer for stochastic gradient descent for the suited properties, our loss function cross-entropy method helps guide our model in a correct direction, and we inform the model that it should use accuracy as its metric as that is what we plan on improving.

We now have our model created, and now need to train the model with the information we have. The neural network is trained, in our case for 25 iterations. Listed are the 1st, 10th, and 25th epoch of our training, describing the loss, and accuracy.

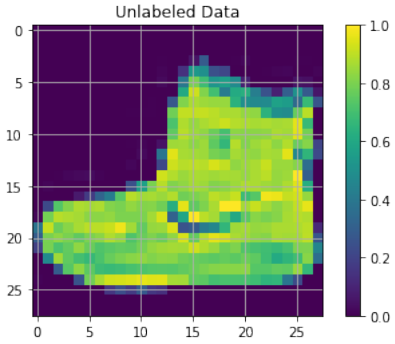
Epoch 1/25
1875/1875 [==============================] - 5s 2ms/step - loss: 0.4989 - accuracy: 0.8232

Epoch 10/25
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2369 - accuracy: 0.9121

Epoch 25/25
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1590 - accuracy: 0.9400

This is great news for our algorithm. After 25 iterations, we predicted the correct article of clothing at around 94 percent. As you perform more and more iterations, your accuracy increases less and less. It should be noted from epoch 1 to epoch 10 there was a greater increase, but from 10 to 25, it was half of what was achieved before.

To validate our results, lets look at the first index of our predictions. As humans, we can tell that these are ankle boots.



Lets see what our algorithm thinks,

- T-Shirt: 2.3181090380575142e-10
- Trousers: 3.5778595364188703e-11
- Pullover: 1.0420617412585167e-13
- Dress: 1.910504835388703e-14
- Coat: 4.149132981634551e-10
- Sandals: 1.7115862647187896e-06
- Shirt: 7.386435907363875e-10
- Sneaker: 6.548938108608127e-05
- Bag: 1.0021435237206333e-08
- Ankle Boots: 0.9999327659606934

Thankfully, our algorithm can detect similar to what us as humans think. This shows accuracy in our pattern. In most cases, there would be more than one test to validate algorithm.

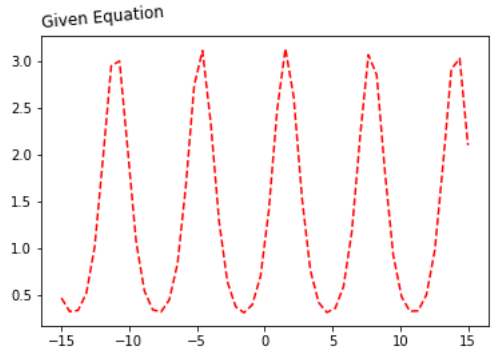
VI. Practice with ML Tools

Although TensorFlow is very helpful, there are other tools required in Machine Learning that are important to understand. A few valuable resources that I will be using in the future are SciPy's API, NumPy's Reference, and a MatPlotLib cheatsheet.

Starting off with graphing, its a quite important part of any math. For simplicity sake, lets graph a simplistic function.

$$y = \pi^{\sin(x)}$$

After referring to the MatPlotLib/Numpy API, the final solution was,



Lets look at the code.

```
x = np.linspace(-15,15)
y = math.pi ** np.sin(x)
plt.plot(x,y, 'r', linestyle = 'dashed')
plt.title("Given Equation", rotation = 5,
    loc = 'left')
plt.savefig('equationimg.png')
```

Listing 5. Generating Graph Based off Equation

VII. Conclusion

Acknowledgment

References