CSC 180-01
Mini-Project 2
Friday October 11 2019

Network Intrusion Detection

By
Warren Quattrocchi (Team Leader) ID# 219714665
Janus Kwan ID# 212910387
Andrew Wright ID# 218771424

(1) Problem Statement

When working with a network, it is desirable to detect unauthorized access. Software to detect network intrusions protects a computer network from unauthorized users, including perhaps insiders. This project aims to build a network intrusion detector, a predictive model capable of distinguishing between bad connections, called intrusions or attacks, and good normal connections. This problem is modeled as a classification problem that determines whether a connection is considered good, or if the connection is an attack, and if so, what type of attack. This project makes use of a Fully-Connected Neural Network model and a Convolutional Neural Network model to detect bad connections (intrusions). It then compares the accuracy, recall, precision and F1-score of each model. It also plots the confusion matrix for each model.

(2) Methodology

This project had two versions. The first version entailed just the basic requirements. The final version contains some elements from beyond the project. The first step to this project was to handle the database provided. Columns were created for each and every data type for each entry along with a column for outcome, which dictated which of the 23 types, different attack types or normal. Then the strings for the three columns, protocol_type, service and flag are one hot encoded, and the rest of the columns are turned into the z-score metric. In the first version after encoding, all attack types are combined into one type, and the rest into normal, this was done to simplify the data. In the final version, each attack type is saved along with normal. In order to avoid troubles, all null values are populated with 0. The data is split into train and test, with a 20% test size. We run this data split into a regressor. Returning to the database it is again split into x,y train and test, with a 25% test size. The x shapes are changed into the appropriate shape. The Model used was sequential, with activation being RELU. 5-7 layers are created, respectively and in this order, 1 convolution layers then a max pooling layer, a layer to flatten the shapes, and two dense layers. The extra two layers not mentioned above are drop out layers which were included to ensure the model does not overfit. Then the accuracy, recall, precision and f1-score, confusion matrix is listed below it. The final part of the project is to run this same database through a fully connected model. Still using sequential and Relu, the database is run through it with a 20% test size.

(3) Experimental Results and Analysis

CNN Parameter Tuning

Conv2D Layers	Activation	Optimizer	Kernel Number	Kernel Size	Dropout Layer	Accuracy
2 layers (64 and 32)	relu	Adam (defaults)	32	(1,3)	.25	0.9979
2 layers (64 and 32)	sigmoid	Adam (defaults)	32	(1,3)	.25	0.6033
2 layers (64 and 32)	tanh	Adam (defaults)	32	(1,3)	.25	0.9974
2 layers (64 and 32)	relu	Adam (defaults)	32	(1,5)	.25	0.9976
2 layers (64 and 32)	relu	Adam (defaults)	64	(1,5)	.25	0.9977
2 layers (64 and 32)	relu	Adam (defaults)	128	(1,5)	.25	0.9977
2 layers (64 and 32)	relu	Adam (defaults)	64	(1,9)	.25	0.9975
1 layer (64)	relu	Adam (defaults)	64	(1,5)	.25	0.9978
1 layer (64)	relu	Adam lr=0.001 beta_1=0.9, beta_2=0.999 decay=1e-6 epsilon=None amsgrad=False	64	(1,5)	.25	0.9977

1 layer (64)	relu	Adam lr=0.001 beta_1=0.9, beta_2=0.999 decay=1e-6 epsilon=None amsgrad=True	64	(1,5)	.25	0.9980
1 layer (64)	relu	Adam lr=0.001 beta_1=0.9, beta_2=0.999 decay=1e-5 epsilon=None amsgrad=True	64	(1,5)	.25	0.9981
1 layer (64)	relu	Adam lr=0.001 beta_1=0.9, beta_2=0.999 decay=1e-3 epsilon=None amsgrad=True	64	(1,5)	.25	0.9984
1 layer (64)	relu	Adam lr=0.001 beta_1=0.9, beta_2=0.999 decay=1e-2 epsilon=None amsgrad=True	64	(1,5)	.25	0.9982
1 layer (64)	relu	SGD lr=0.001 decay=1e-6 momentum = 0.9 epsilon=None Nesterov = True	64	(1,5)	.25	0.9977
1 layer (64)	relu	SGD lr=0.001 decay=1e-6 momentum = 0.9 epsilon=None Nesterov = True	64	(1,5)	.25	

1 layer (64)	relu	SGD lr=0.001 decay=1e-6 momentum = 0.9 epsilon=None Nesterov = False	64	(1,5)	.25	0.9957
1 layer (64)	relu	SGD lr=0.001 decay=1e-5 momentum = 0.9 epsilon=None Nesterov = False	64	(1,5)	.25	0.9979

Relu had the best results out of the activations, with Tanh having a slightly worse accuracy and Sigmoid generating results that seemed incorrect (model guessed 'normal' on all test sets. The accuracy was slightly better with one Conv2D layer. Adam had the best results for optimizers, with SGD coming close with a certain set of parameters(lr=0.001, decay = 1e^-6, momentum = 0.9, epsilon = None, Nesterov = False) but Adam had more consistently better results. A kernel size of (1,5) provided the best accuracy with raising or lowering causing a small loss. Tampering with the dropout layer didn't seem to cause notable changes by being lowered or raised on our best CNN model.

Fully Connected Model Parameter Tuning

Layer(s)	Activation	Optimizer	min_delta	Accuracy	F1-Score
Dense(50) Dense(25)	‘relu’	Adam	1e-5	1.00	1.00
Dense(100) Dense(50)	‘relu’	Adam	1e-5	1.00	1.00
Dense(50) Dense(25)	‘relu’	SGD	1e-5	1.00	1.00
Dense(50) Dense(25)	‘relu’	Adam	1e-2	1.00	1.00
Dense(50) Dense(25)	‘tanh’	Adam	1e-2	1.00	1.00
Dense(50) Dense(25)	‘tanh’	SGD	1e-2	0.99	0.99

(4) Task Division and Project Reflection

Similarly to Project 1, we created an initial working model of the CNN to make sure we could get everything working before tampering with the data to get more accurate results or creating the fully-connected network. After ensuring that our CNN was outputting acceptable results we started working with the data to get a better F1-score. The task division for Project 2 followed a similar template to Project 1, with each of us spending more time on a certain aspect of the project, such as Janus with the fully-connected model, Warren with the CNN model, and Andrew with parameter tuning.

One of the problems we encountered during this project were dealing with N/A values in the dataset. Running the regressor before cleaning the data resulted in an error, and training the model resulted in inaccurate data predicting ‘normal’ for every item in the test set. This was fixed because it turned out the code we used to remove the N/A values was applying to the wrong table, resulting in no action being performed on that table.