KDD ‘99 Dataset Neural Network & its Feature Selection

Kaihua Cai
Weiqi Yao
Kc3172 & wy697

Outline

1. Introduction
    A. Problem Statement & Objectives
    B. KDD' 99 dataset
    C. Project Method
2. Review of Past techniques
3. Evaluation Results
    A. Data Preprocessing
    B. Neural Network Model
    C. Results of Non-PCA
    D. Results of PCA
    E. Results of borrowed feature selection
    F. Results Comparison
4. References

1. Introduction: Problem Statement & Objective

- Intrusion detection, a common network security research problem:
  - Its aim: build a predictive model (i.e. a classifier) capable of distinguishing between “bad” connections and “good”.
  - “bad” connections, intrusions or attacks generated from hackers,
  - “good” connections, normal traffic generated from normal users.

- An ideal intrusion detection system should be able to detect intrusions from hackers ***fast and precisely*** (network line-rate)
  - i.e. the administrator is informed of attacks the moment it reaches the network, or more formerly, detection at line-rate.

1. Introduction: Problem Statement & Objective

- However, due to the mass size of the data to be processed in real time network, most intrusion detection system right now is offline.

- Objective: ***a simple and rather precise*** machine learning classifier that is possible to deploy online.

1. Introduction: KDD ’99 Dataset

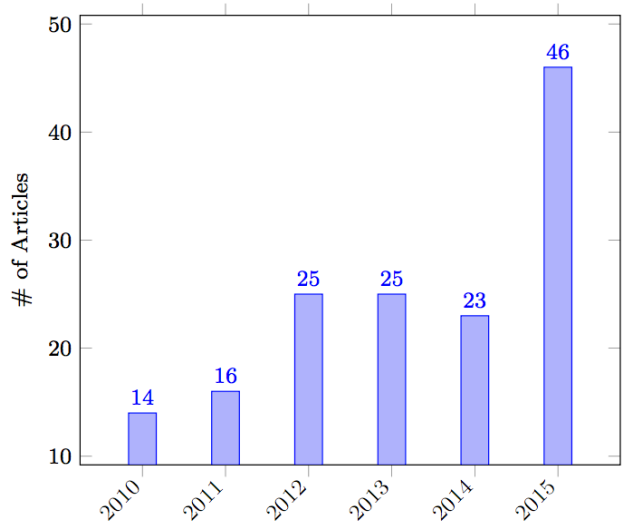
- KDD ‘99 Dataset
  - A competition held by KDD in 1999, its task was to build a network intrusion detector to distinguish between ”bad” (intrusions/attacks) and “good” normal connections.
  - Its trace is a wide variety of intrusions simulated in a military network environment.
    - Lincoln Labs acquired 9 weeks of raw TCP dump data(DARPA-sponsored IDS-event) for a local-area network (LAN) simulating a typical U.S. Air Force LAN, and mixed with multiple attacks.
    - Lee and Stolfo, one of the participating teams of the DARPA event, gave their feature extracted and preprocessed data to Knowledge Discovery and Data Mining (KDD) yearly competition.
    - To reduce deficiencies of KDD99 dataset for machine learning algorithm, Tavallaee et al. introduced NSL-KDD dataset.

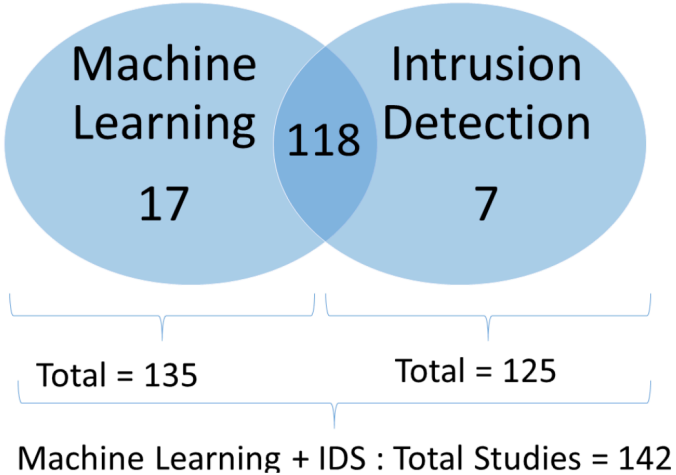
1. Introduction: KDD ’99 Dataset

- KDD ‘99 Dataset(contd.)
  - The KDD CUP 99 training data set consists of approximately 4,900,000 samples with 41 features and each sample is labeled as either normal or attack.
  - The features in KDD CUP 99 Dataset are divided into 3 sets:
    - Basic features of individual TCP connections, e.g. *duration*;
    - Content features within a connection suggested by domain knowledge, e.g. *num_file_creations*;
    - Traffic features computed using a two-second time window, e.g. *same_srv_rate*;
    - Statistical characteristics of network traffic based on host(100 connections), e.g. *dst_host_count*.
  - There are a total of 24 attacks, which fall into 4 main categories:
    - DOS: denial-of-service, e.g. *SYN flood*;
    - R2L: unauthorized access from a remote machine, e.g. *guessing password*;
    - U2R: unauthorized access to local super-user (root) privileges, e.g. *various ``buffer overflow'' attacks*;
    - probing: surveillance and other probing, e.g., *port scanning*.

1. Introduction: KDD ’99 Dataset

- KDD ‘99 Dataset(contd.)
  - Total of 142 articles use KDD CUP 99 in either Machine Learning or IDS between 2010 and 2015 from 149 research articles





1. Introduction: Project Method

1. Use neural network as a classifier model;
   - Provide a high true positive and true negative detection rate;
   - But compared with other classify methods, neural network will have larger training and validation time. Contradictory to our objective in keeping up with network line-rate;
   - Methods like: Clustering, C4.5 decision tree, SVM, K-means and K Nearest Neighbor.

2. Use PCA as *Feature Selection* to downsize the number of inputs, so to compensate for the decrease in detection speed.
   - Convert 120 features to only 20 principle components.

3. Compare the result of PCA with neural network that has all features as input.

4. Compare again with features selected from a recent paper [3]. Apply the features selected there to our own model.

2. Review of Past techniques

- K-means clustering in Spark, from the book *Advanced analytics with Spark: patterns for learning from data at scale* (pp. 81-97)
  - This provide detailed information on how to build a classifier for KDD ‘99 dataset, however the techniques used here is K-means clustering and has a rather low accuracy in validation, around 91%.

- J. (n.d.). Jeffheaton/t81_558_deep_learning. https://github.com/jeffheaton/t81_558_deep_learning/blob/master/tf_kdd99.ipynb
  - A github code from deep-learning course in Washington University of Saint Louis.
  - This provide the basic code for building a simple neural network on KDD ’99.
  - We borrowed its code for one-hot encoding and normalization of the dataset as well as the basic neural network model.

3. Evaluation Results: Data Preprocessing

- Data Preprocessing
  - We start by working on a reduced dataset (10% dataset), where the dataset is split by 75% for training and 25% for testing.
  - The 10%dataset will be preprocessed to do feature scaling(normalization) and one-hot encoding, resulting in a transformation of 42 features to 120 features.
  - The model will be validated on the 10% testing dataset and also the complete dataset.

Dataset before Preprocess

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...
0	0	tcp	http	SF	181	5450	0	0	0	0	...
1	0	tcp	http	SF	239	486	0	0	0	0	...
2	0	tcp	http	SF	235	1337	0	0	0	0	...
3	0	tcp	http	SF	219	1337	0	0	0	0	...
4	0	tcp	http	SF	217	2032	0	0	0	0	...

5 rows × 42 columns

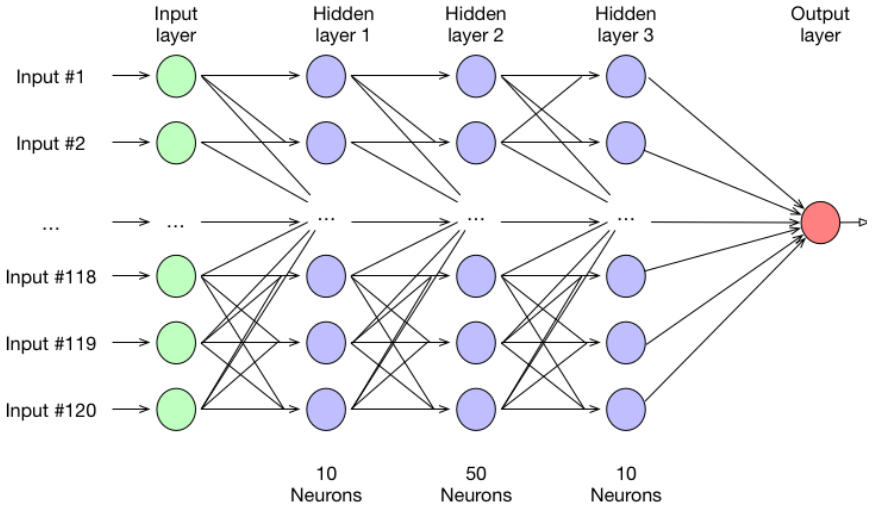
Dataset after Preprocess

	duration	src_bytes	dst_bytes	wrong_fragment	urgent	hot	num_failed_logins
0	-0.067792	-0.002879	0.138664	-0.04772	-0.002571	-0.044136	-0.009782
1	-0.067792	-0.002820	-0.011578	-0.04772	-0.002571	-0.044136	-0.009782
2	-0.067792	-0.002824	0.014179	-0.04772	-0.002571	-0.044136	-0.009782
3	-0.067792	-0.002840	0.014179	-0.04772	-0.002571	-0.044136	-0.009782
4	-0.067792	-0.002842	0.035214	-0.04772	-0.002571	-0.044136	-0.009782

5 rows x 120 columns

3. Evaluation Results: Neural Network Model

- Neural Network Classifier
  - Then build a simple deep neural network model with 3 hidden layers, to classify/predict entries into 2 classes(attacks and normal).



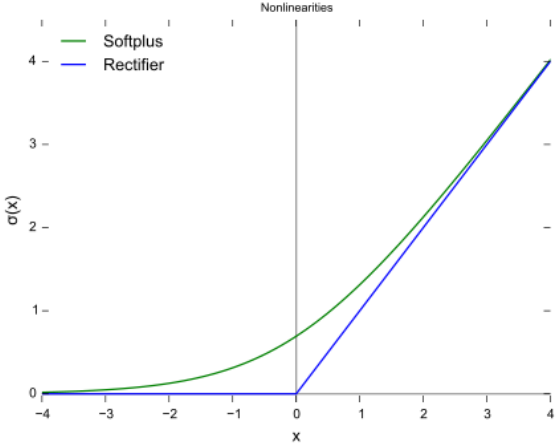
Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 10)	1210
dense_2 (Dense)	(None, 50)	550
dense_3 (Dense)	(None, 10)	510
dense_4 (Dense)	(None, 1)	11
dense_5 (Dense)	(None, 1)	2

Total params: 2,283
Trainable params: 2,283
Non-trainable params: 0

3. Evaluation Results: Neural Network Model

- Neural Network Training Parameters
  - Uses ‘ReLU’ (rectified linear unit) activation function to speed up training process.
  - The output is a logistic activation function, which is used for various binary class classification methods.
  - Loss function is “categorical_crossentropy”, $L(\hat{y}, y) = \frac{1}{N}\sum_i^N [y_i \cdot log \frac{1}{\hat{y}_i}]$.
  - Gradient descent learning rate is initially 0.001.
  - Batch size is set to 100 and epoch set to 5.

ReLU activation function




```
from keras import optimizers

# optimizer defined
opt = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
```

```
## compile model
model.compile(optimizer=opt,
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

3. Evaluation Results: Results of Non-PCA

- Training and validation results for no PCA set

```
Train on 370515 samples, validate on 123506 samples
Epoch 1/5
370515/370515 [==============================] - 13s - loss: 0.0206
- acc: 0.9951 - val_loss: 0.0048 - val_acc: 0.9982
Epoch 2/5
370515/370515 [==============================] - 16s - loss: 0.0039
- acc: 0.9987 - val_loss: 0.0034 - val_acc: 0.9990
Epoch 3/5
370515/370515 [==============================] - 18s - loss: 0.0032
- acc: 0.9992 - val_loss: 0.0033 - val_acc: 0.9990
Epoch 4/5
370515/370515 [==============================] - 22s - loss: 0.0027
- acc: 0.9993 - val_loss: 0.0030 - val_acc: 0.9992
Epoch 5/5
370515/370515 [==============================] - 21s - loss: 0.0026
- acc: 0.9993 - val_loss: 0.0027 - val_acc: 0.9993
```

The loss function (categorical crossentropy) value in the training dataset for the last batch in this epoch.

The categorical accuracy in the training dataset for the last batch in this epoch. (Correct accuracy)
Acc = (argmax(y_true) == argmax(y_pred))

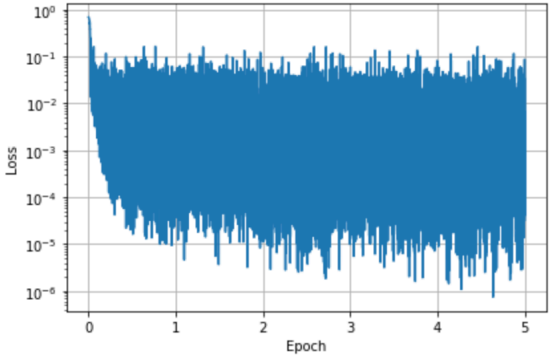
The loss function value of the testing dataset for the model trained at the end of this epoch

The categorical accuracy of the testing dataset for the model trained at the end of this epoch

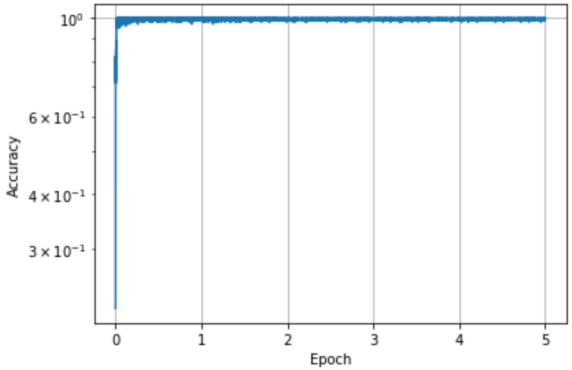
3. Evaluation Results: Results of Non-PCA

- Training and validation results (contd.)
  - Loss function value is recorded at the end of each batch, the loss value for training dataset is infinitely approximating the optimal value.
  - While validation accuracy and validation loss is only calculated at every epoch's end.

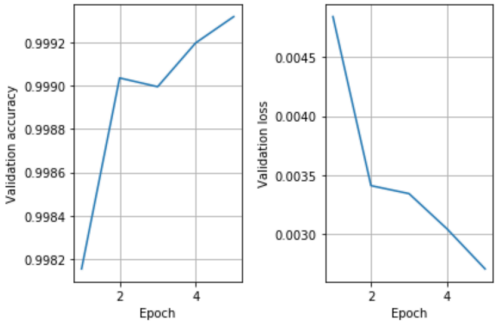
Training Loss



Training Accuracy



Testing Loss and Accuracy



3. Evaluation Results: Results of Non-PCA

- Validation on whole KDD '99 dataset, instead of 10%
  - Time to validate is approximately 240s

```
print("Accuracy is: %s" %(1-count/yhat_size))
print("# of wrongly classified: %s" %(count))
```

```
Accuracy is: 0.9994051156380482
# of wrongly classified: 2914
```

```
print(classification_report(Y_whole, y_pred_whole))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00    972781
          1       1.00      1.00      1.00   3925650

avg / total       1.00      1.00      1.00   4898431
```

```
Cm = confusion_matrix(Y_whole,y_pred_whole)
C = np.sum(Cm)
Cm = Cm/C
print('Confusion Matrix:')
print(np.array_str(Cm, precision=4, suppress_small=True))
```

```
Confusion Matrix:
[[ 0.1982  0.0004]
 [ 0.0002  0.8012]]
```

3. Evaluation Results: Results of PCA

- PCA performed for both training and validation
  - Training dataset, from 120 features to 20 features

```
# ######################
# #########PCA##########
# ######################

# # feature extraction
pca = PCA(n_components=20)
fit = pca.fit(X.T)

# summarize components
print(("Explained Variance: %s") % fit.explained_variance_ratio_)
print(np.shape(fit.components_.T))
X = fit.components_.T
# print(fit.components_)
```

```
Explained Variance: [ 0.22845019  0.12651505  0.11281686  0.07454234  0.0
6876901  0.03971399
  0.02952583  0.02902571  0.02674098  0.02584283  0.02534576  0.02521773
  0.02452765  0.02296654  0.02176682  0.02160914  0.01961439  0.01834038
  0.01812486  0.00973112]
(494021, 20)
```

The variance of the 20 Principle Components, ranged from largest to smallest

The number of features now is 20

3. Evaluation Results: Results of PCA

- PCA performed for both training and validation
  - Testing dataset, from 120 features to 20 features

```
fit_whole = pca.fit(X_whole.T)
print(("Explained Variance: %s") % fit_whole.explained_variance_ratio_)
print(np.shape(fit_whole.components_.T))
X_whole = fit_whole.components_.T
```

```
Explained Variance: [ 0.22962152  0.1264887   0.11289027  0.07114857  0.0
5875115  0.03964593
  0.03134808  0.02642499  0.02582551  0.02562436  0.0254813   0.0254513
  0.02535263  0.0249264   0.02399793  0.02320513  0.02133873  0.02096531
  0.018345    0.012523  ]
(4898431, 20)
```

The variance of the 20 Principle Components, ranged from largest to smallest

  - The testing dataset variance is quite similar to the training

3. Evaluation Results: Results of PCA

- Training and validation results for PCA set

```
Train on 370515 samples, validate on 123506 samples
Epoch 1/5
370515/370515 [==============================] - 12s - loss: 0.2653
- acc: 0.8971 - val_loss: 0.0405 - val_acc: 0.9891
Epoch 2/5
370515/370515 [==============================] - 12s - loss: 0.0395
- acc: 0.9903 - val_loss: 0.0369 - val_acc: 0.9915
Epoch 3/5
370515/370515 [==============================] - 12s - loss: 0.0376
- acc: 0.9910 - val_loss: 0.0362 - val_acc: 0.9915
Epoch 4/5
370515/370515 [==============================] - 12s - loss: 0.0369
- acc: 0.9912 - val_loss: 0.0356 - val_acc: 0.9916
Epoch 5/5
370515/370515 [==============================] - 12s - loss: 0.0364
- acc: 0.9913 - val_loss: 0.0351 - val_acc: 0.9917
```

The loss value increased and the accuracy dropped in the 1st epoch, compared to the results of non-PCA.

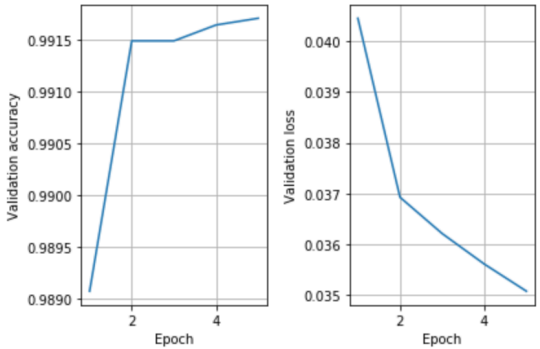
The Validation loss and accuracy is not as ideal as the results from non-PCA, but still in acceptable range.

- This result is expected because features from 120 to 20 means drop in information. Thus reducing Accuracy.

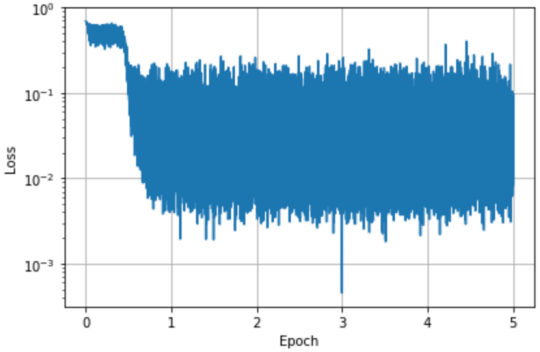
3. Evaluation Results: Results of PCA

- Training and validation results (contd.)

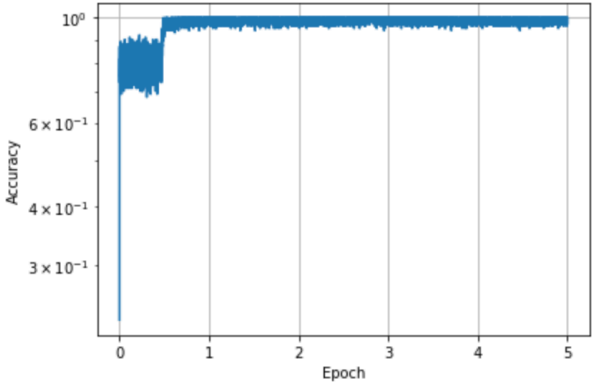
Testing Loss and Accuracy



Training Loss



Training Accuracy



3. Evaluation Results: Results of PCA

- Validation on whole KDD '99 dataset, instead of 10%
  - Time to validate is approximately 90s, time to validate greatly reduced!

```
print("Accuracy is: %s" %(1-count/yhat_size))
print("# of wrongly classified: %s" %(count))
```

```
Accuracy is: 0.8005046922167527
# of wrongly classified: 977214
```

```
print(classification_report(Y_whole, y_pred_whole))
```

```
             precision    recall  f1-score   support

          0       0.45      0.02      0.04    972781
          1       0.80      0.99      0.89   3925650

avg / total       0.73      0.80      0.72   4898431
```

```
Cm = confusion_matrix(Y_whole,y_pred_whole)
C = np.sum(Cm)
Cm = Cm/C
print('Confusion Matrix:')
print(np.array_str(Cm, precision=4, suppress_small=True))
```

```
Confusion Matrix:
[[ 0.0042  0.1944]
 [ 0.0051  0.7963]]
```

A significant drop in accuracy in the complete dataset. From 99.94% in non-PCA validation to only 80.05%.

This is probably due to under-fitting of the model, and also great change of data pattern in training 10% dataset and testing 100% dataset.

3. Evaluation Results: Results of borrowed features

- From the paper, Relevance feature selection with data cleaning for intrusion detection system, which uses an information gain method to do feature selection.
- Select only the features stated there in the normal class label

- Then our features will downsize from 41 features to 11 features before one-hot encoding
- After one-hot encoding, from 120 features to 87 features.

Table 4. List of features for which the class is selected most relevant

Class Label	Relevant Features
normal	1, 6, 12, 15, 16, 17, 18, 19, 31, 32, 37
smurf	2, 3, 5, 23, 24, 27, 28, 36, 40, 41
neptune	4, 25, 26, 29, 30, 33, 34, 35, 38, 39
land	7
teardrop	8
ftp_write	9
back	10, 13
guess_pwd	11
buffer_overflow	14
warezclient	22

```
df_new = df[['service',
     'flag',
     'src_bytes',
     'dst_bytes',
     'land',
     'num_root',
     'rerror_rate',
     'diff_srv_rate',
     'dst_host_diff_srv_rate',
     'dst_host_srv_serror_rate',
     'outcome']]
```

3. Evaluation Results: Results of borrowed features

- Training and validation results for borrowed feature set

```
Train on 370515 samples, validate on 123506 samples
Epoch 1/5
370515/370515 [==============================] - 15s - loss: 0.0580
- acc: 0.9841 - val_loss: 0.0239 - val_acc: 0.9913
Epoch 2/5
370515/370515 [==============================] - 15s - loss: 0.0219
- acc: 0.9946 - val_loss: 0.0195 - val_acc: 0.9949
Epoch 3/5
370515/370515 [==============================] - 15s - loss: 0.0198
- acc: 0.9950 - val_loss: 0.0189 - val_acc: 0.9949
Epoch 4/5
370515/370515 [==============================] - 15s - loss: 0.0188
- acc: 0.9950 - val_loss: 0.0174 - val_acc: 0.9958
Epoch 5/5
370515/370515 [==============================] - 14s - loss: 0.0179
- acc: 0.9954 - val_loss: 0.0180 - val_acc: 0.9947
```

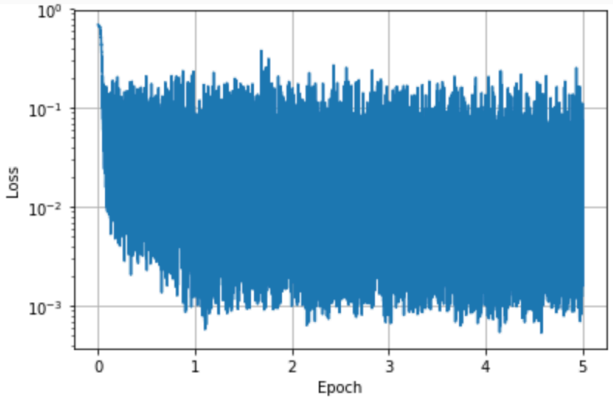
Compared to the results of non-PCA and PCA, its training loss and accuracy is in between them.

The loss function value and accuracy of the testing dataset is in between as well. But closer to the full dataset.

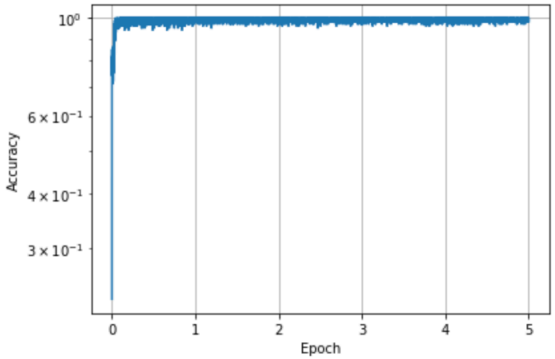
3. Evaluation Results: Results of borrowed features

- Training and validation results (contd.)

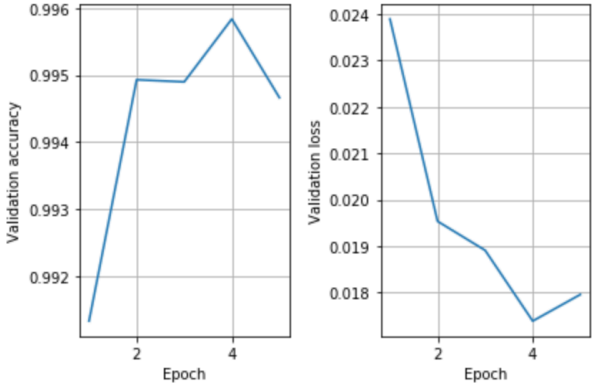
Testing Loss and Accuracy



Training Loss



Training Accuracy



3. Evaluation Results: Results of borrowed features

- Validation on whole KDD '99 dataset, instead of 10%
  - Time to validate is approximately 90s, same as the PCA set

```
print("Accuracy is: %s" %(1-count/yhat_size))
print("# of wrongly classified: %s" %(count))
```

```
Accuracy is: 0.9977035503817446
# of wrongly classified: 11249
```

```
print(classification_report(Y_whole, y_pred_whole))
```

```
              precision    recall  f1-score   support

          0       1.00      0.99      0.99    972781
          1       1.00      1.00      1.00   3925650

avg / total       1.00      1.00      1.00   4898431
```

```
Cm = confusion_matrix(Y_whole,y_pred_whole)
C = np.sum(Cm)
Cm = Cm/C
print('Confusion Matrix:')
print(np.array_str(Cm, precision=4, suppress_small=True))
```

```
Confusion Matrix:
[[ 0.1969  0.0017]
 [ 0.0006  0.8008]]
```

A very close result to the non-PCA, all features as input set.
But uses half the time than the non-PCA set in validation of 4898431 rows of data while still maintaining high accuracy in detection.

OUR OBJECTIVE IS MET!!

3. Evaluation Results: Results Comparison

- Validation Time:
  - For the non-PCA dataset, the validation time for classifying data on all 4,898,431 data is 240s.
  - While PCA set and borrowed-features set uses only 90 s.
- Validation Accuracy:
  - For the PCA set, its accuracy is 0.8005, this is probably due to the underfitting of data.
  - While for the non-PCA set and the borrowed-features set, the accuracy is >=0.9977

- So our objective is met when using the borrowed-features set, with low validation time yet high detection rate.

4. References

- J. (n.d.). Jeffheaton/t81_558_deep_learning. Retrieved December 16, 2017, from https://github.com/jeffheaton/t81_558_deep_learning/blob/master/tf_kdd99.ipynb
- K-means clustering in Spark. In S. Ryza, U. Laserson, S. Owen, & J. Wills (Authors), *Advanced analytics with Spark: patterns for learning from data at scale* (pp. 81-97). Sebastopol, CA: OReilly.
- Suthaharan, S., & Panchagnula, T. (2012). Relevance feature selection with data cleaning for intrusion detection system. *2012 Proceedings of IEEE Southeastcon*. doi:10.1109/secon.2012.6196965