# 1 Abstract

Network intrusion detection system (NIDS) is essential to the security of any systems with communication capacity. Recently a handful of novel deep neural networks bundled with more advanced and smarter training algorithms have achieved unprecedentedly good performance on image classification, natural language processing, speech recognition and many other research branches. Motivated by these impressive improvements in the field of artificial intelligence, this paper tries to answer the following questions: Can we transfer hall-of-fame deep learning approaches to network intrusion detection task? If yes, how much improvement can be expected? If no, what are the reasons?

We answer these questions in four steps. Firstly, we introduce deep learning models and techniques and why they may better solve the network intrusion detection problem. Then we briefly review the existing machine learning solutions to network intrusion detection, some of which provide the state-of-the-art detection performance. After that, we describe several groups of the cutting-edge deep learning models in concisely mathematical languages. We conduct a quantitatively comparative study of each of them with two off-line network intrusion detection datasets, with the help of our own TensorFlow-based deep learning library, NetLeaner. Apart from making NetLearner publicly available, we also share the hacks and tricks used during the training phase so that future researchers can easily reproduce and extend our work.

# 2 Introduction

Network intrusion detection system (NIDS) is the essential security technology that aims to protect a computer network intelligently and automatically. As either a hardware device or software application, it monitors a network for malicious activities or policy violations. By intercepting and analyzing the bi-direction traffics through the network, it raises alarm if intrusion, attack or violation are observed. There are two general approaches to detect intrusions. In signature based intrusion detection, e.g. SNORT [4], rules for specific attacks are pre-installed in the system. It report suspicious traffic when the traffic matches any signature of known attacks. The major drawback of signature matching approach is that it is only effective for previously detected attacks that have an identifiable signature. As a result, signature database needs to be manually updated whenever a new type of attack is discovered, with significant effort, by the network administrator. Anomaly detection based approach overcomes these limitations by adopting a certain type of machine learning technique to model the trustworthy network activities. Traffics that significantly deviates from the built model are treated as malicious. This idea have been shown to be able to detect unknown or novel attacks [19, 37]. However, if the built model for normal traffics are not generalized enough, anomaly based approach will treat unforeseen normal traffic as malicious, suffering from high false positive.

In this project, we follow the anomaly detection based idea, and tries to enhance it with the state-of-art machine learning technology, e.g. various deep learning architectures. Specifically, we have made the following contributions:

- Firstly, we introduce the background of deep learning models and techniques, and discuss why they may better solve the network intrusion detection problem.
- Then we briefly review the existing state-of-the-art machine learning solutions to network intrusion detection. After that, we describe several groups of the cutting-edge deep learning models in concisely mathematical languages.
- We conduct a quantitatively comparative study of each of them with two off-line network intrusion detection datasets [26, 37], with the help of our own TensorFlow-based deep learning library, NetLeaner. The detection performance is measured in accuracy, precision, recall and F-Score, with detailed confusion matrix.

We not only make the codebase of NetLearner publicly available to research community, but also share the deep learning related hacks and tricks used during the training phase, so that future researches can easily reproduce and extend our work.

## 3 Deep Learning Background

We identify three main reasons why deep learning succeed in many areas related to artificial intelligence, as well as their implications on network intrusion detection problem.

### 3.1 Learning/Training Techniques

In the supervise learning framework, given the feature representations and inference model, learning is in essential an optimization problem which minimizes a predefined loss function over given training examples. The most commonly used optimization algorithm is back-propagation(BP) [32] with gradient descent, because computing gradient is Hessian-free and memoization saves a great amount of computation when propagating backward level by level. However, it is impossible to train deep neural networks and achieve optimal parameters only with BP. The first problem is that usually the cost function is non-convex with a lot of local minima; optimizing algorithms using only first-order gradient is likely to be stuck at a poor local minimum. Secondly, exploding and vanishing gradient makes back-propagation difficult to train models with many layers stacked together, such as recurrent neural networks and stacked Restricted Boltzmann Machines. Even if we can tolerate the long training time and carefully deal with gradient exploding and vanishing, the trained model is usually over-fitted to the training dataset, and not able to generalize well to the testing or future dataset.

The emergence of many novel learning algorithms and training techniques makes training deep neural network and achieving good suboptimal minimum possible. For example, stochastic gradient descent (SGD) with mini-batches can greatly increase the training speed comparing to normal gradient descent on the entire dataset. In each step of gradient descent, researchers have shown that momentum [35] can prevent SGD from oscillating across but pushing along the shallow ravine. Along with decaying learning rate, momentum-based optimization algorithms, for example Adam [20], usually help us find better local minimum. To prevent over-fitting, researchers have proposed dropout [34] to average over exponential number of neural networks. These learning algorithms and training techniques will directly help neural networks achieve better performance for the network intrusion detection problem, since they are general to any types of neural network models.

### 3.2 Unsupervised Generative Models

Another breakthrough in the deep learning area is that researchers have successfully trained a number of unsupervised generative models that attracted much attentions. Different from supervised models (or discriminative models) that tries to discover the relationship between input variables and target label (or the conditional probability distribution of the targets given the input), these models aims to learn the joint probability distribution, or joint conditional distribution, of **all** variables for a phenomenon from the given dataset. The resulting generative model is powerful in many ways. First, given the well trained probability distribution, the model can synthesize meaningful data comparable to real examples in training set. For example, Auxiliary-Classifier Generative Adversarial Nets (AC-GAN) [29] can generate very high quality images after training on ImageNet dataset [33]; both stacked denoising autoencoder [38] and deep brief nets [16] can synthesize handwritten digits after learning from the MNIST dataset. Besides, the ability to generate meaningful and high quality faked data actually means that the model have learned better feature representations from the unlabeled data itself. As an example, it is shown that the features extracted from the hidden units

Table 1: Popular Datasets used in Deep Learning v.s. Available Network Intrusion Detection Datasets

| Domain | Dataset | Training Examples | Feature Dimension |
|---|---|---:|---:|
| Image | MNIST | 60,000 | 784 |
| | SVHN | 600,000 | 3072 |
| | CIFAR-10 | 60,000 | 3072 |
| | Tiny | 80 million | 3072 |
| | ImageNet | 1.2 million | 196,608 |
| IDS | UNSW-NB15 | 175,341 | 42 |
| | NSL-KDD | 125,973 | 41 |

of sparse autoencoder can significantly improve the performance of support vector classifier [31]. At last, researchers have shown that it is usually a good strategy to initialize deep neural networks with the weights from a successfully trained generative model [16, 35].

In the area of network security, the amount of network traffic data is enormously large, usually in the order of terabytes per day in a large monitored network. In practice, the amount of data is impossible for a human security analyst or a group of them to review, e.g., to find patterns and label anomalies. This situation makes an unsupervised generative model a promisingly good solution to traffic classification since it can be trained unsupervised:

- It utilizes the large amount of unlabeled data to learning useful and hierarchical features from the traffic data itself;
- It is equivalently a good way to initialize the weights of the hidden layers in a deep neural network, which can be further fine-tuned to be a high performance classifier.

In this project we investigate and try three types of generative models: restricted Boltzmann machine, autoencoders and generative adversarial nets. See section 5 for details.

### 3.3 Datasets

Apart from the sophisticated and efficient learning algorithms, abundant open data in the domain of image classification, natural language processing, machine translation, etc. actually drives the novel and complex neural network models and make them successful.

The most vivid example would be how ImageNet dataset [33] pushed a series of deep learning models, such as AlexNet, GoolgNet and ResNet, who greatly improved the performance of visual object recognition. ImageNet organizes a large amount of web images by synonym set, multiple words or word phrases describing a meaningful concept. On average each synonym set is illustrated by 1000 quality-controlled and human-annotated images. This project was first presented at 2009 Conference on Computer Vision and Pattern Recognition by researchers from the CS department at Princeton University, and ran as an annual software contest known as ImageNet Large Scale Visual Recognition Challenge (ILSVRC) since 2010. The state-of-the-art error rate of this competition was near 25%, until in the year of 2012, a deep convolutional neural networks AlexNet [22] trained on GPU achieved a winning top-5 error rate of 15.3%. From then on, deep neural networks with convolutional blocks start its showtime. GooLeNet [36] won the ILSVRC 2014 with top-5 error rate of 6.7%. It has 22 layers and strayed from the simple design of stacking convolutional and pooling layers. One year later, residual block based ResNet [13] pushed the error rate further down to 3.6% with even deeper architecture.

# 4 Existing Works

Due to the large number of network intrusion detection systems that adopt machine learning or data mining approaches we only review a few of them that achieve state-of-the-art detection performance. There are very limited number of existing network intrusion detection systems that adopt deep learning approaches. Their details can be found in section 5.

## 4.1 State-of-Art Machine Learning Approaches

Prior researchers modeled the intrusion detection task as an unsupervised anomaly detection problem, and proposed a series of approaches. Examples include Mahalanobis-distance based outliner detection [23], density-based outliner detection [7, 23], evidence accumulation for ranking outliner [8], etc. One of the advantage of these unsupervised approaches is to tackle the problem of the unavailability of labeled traffic data.

Alternatively, prior researchers made a lot of effort to obtain meaningful attacking data and to convert them into labeled data [2, 25–27, 37]. Such efforts make it possible to apply supervised machine learning algorithms to the intrusion detection problem. Successfully applied approaches include decision trees [30], linear and non-linear support vector machines [10], NB-Tree [21] and so on. To the best of the authors' knowledge, there are two works that achieved the best prediction accuracy on the two different datasets respectively. For the UNSW-NB15 dataset, it is reported in [17] that extending K-support vector classification-regression [6] with ramp loss, called Ramp-KSVCR approach, can achieve the state-of-the-art accuracy of 93.52%. The authors of Ramp-KSVCR also report that their approach can achieve 98.68% accuracy on the NSL-KDD dataset. On the other hand, the creators of UNSW-NB15 dataset [26] proposed an approach called Geometric Area Analysis techniques using trapezoidal area estimation (GAA-ADS for short) [28]. It achieves the bese known accuracy on NSL-KDD dataset (99.7%) and a slightly worse accuracy on UNSW-NB15 dataset (92.8%).

## 4.2 Deep Learning Flavor Approaches

There are some pioneer works that introduced deep learning approaches to intrusion detection. For example, [19] adopts sparse autoencoder and the self-taught learning scheme [31] to handle the problem of limited amount of labeled data for training supervised model. Similar semi-supervised approach have also been applied to Discriminative restricted Boltzmann machine [11].

# 5 Deep Learning Architectures

In this section, we give mathematical reviews of the deep learning architectures that we consider promising in the network intrusion detection problem.

## 5.1 Multilayer Perceptron

Multilayer perceptron (MLP) is a classic deep learning classifier with simple design of the connectivity between neurons. It is a fully connected feed-forward neural network, as shown in Figure 1. By introducing non-linear neural units (perceptrons), it can distinguish data that are not linearly separable. However, the non-linearity make it very hard to train a deep MLP of more than three layers, even if people have proposed the efficient back-propagation learning algorithm [32]. Recently it revived due to the various new training techniques designed by deep learning community, including Stochastic Gradient Descent (SGD), batch normalization [18] and Dropout [34]. Except for the number of neurons in each layer and number of layers, MPL can also be tuned with different activation functions, or neural types. The most popular two, which are used in this project, are logistic function and rectifier linear unit. Logistic function is written as

$$f(x) = \frac{1}{1+e^{-x}} \tag{1}$$

It has a very useful property when we applying back-propagation:

$$f'(x) = f(x)(1 - f(x)) \tag{2}$$

Recently, most deep neural networks adopt rectifier neural unit and achieved very good performance [24]. Rectifier linear unit is defined as

$$f(x) = \max(0, x) \tag{3}$$

Let $\mathbf{a}^{(l)}$ be the activation of layer $l$, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ be layer $l$'s parameter. With activation function defined, we have the following recursive formula that describes the feed-forward step of the perceptron network.

$$\mathbf{z}^{(l+1)} = \mathbf{W}^{(l)}\mathbf{a}^{(l)} + \mathbf{b}^{(l)} \tag{4}$$

$$\mathbf{a}^{(l+1)} = f(\mathbf{z}^{(l+1)}) \tag{5}$$

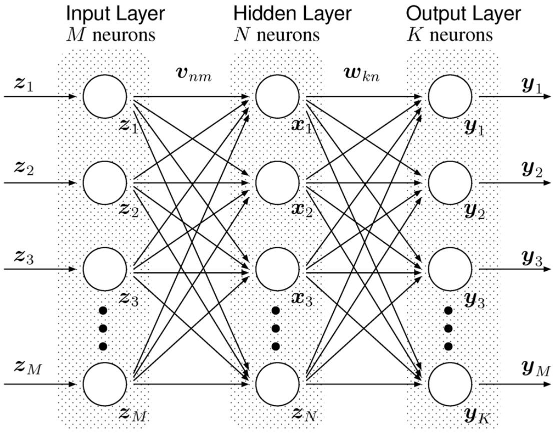

Figure 1: A multilayer perceptron neural network with 1 hidden layer. Figure courtesy of Teijiro Isokawa, Haruhiko Nishimura and Nobuyuki Matsui.

## 5.2 Restricted Boltzmann Machine

Restricted Boltzmann machine (RBM) [14] is a type of energy-based model, which associate a scalar energy to each configuration vector of the variables in the network. In energy-based model, learning is the process of configuring the network weights so that the average energy over training data is minimized. RBM consists of a layer of hidden units (H) and a layer of visible units (V). Here "restricted" means that connections are just between hidden and visible layer, but not within hidden layers or visible layers. This makes its training to be faster than Boltzmann machine and makes it feasible to stack multiple separately trained RBM together to form deep architecture. A joint configuration, $(\mathbf{v}, \mathbf{h})$, of the visible and hidden units has the energy of

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \tag{6}$$

where $a = \{a_i\}$ and $b = \{b_j\}$ are biases in visible and hidden layer respectively, and $W = \{w_{ij}\}$ is the weights between them. The network assigns a probability to every possible pair of $(\mathbf{v}, \mathbf{h})$ via this energy function

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})} \tag{7}$$

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{8}$$

where $Z$ is the partition function that equals to the summation over all possible hidden and visible vector pairs

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{9}$$

Based on the "maximizing log likelihood" idea, we want to raise the probability of a training example and it can be done by adjusting the weights biases to lower the energy of the considered example. Meanwhile, we can let other examples make a big contribution to the partition function $Z$ by raising their energy. Both insights can be translated to the following formula:

$$\frac{\partial \log p(v)}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{10}$$

This implies the following learning rule for performing stochastic gradient ascent on training data

$$\Delta w_{ij} = \varepsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}) \tag{11}$$

The first term $\langle v_i h_j \rangle_{data}$ is the sampling from the data and it is easy to compute since there is no directed connection between hidden units. The sampling of $h_j$ is based on the probability

$$Prob(h_j = 1|\mathbf{v}) = sigmoid(b_j + \sum_i v_i w_{ij}) \tag{12}$$

Similarly, $v_i$ can be sampled with the following distribution

$$Prob(v_i = 1|\mathbf{h}) = sigmoid(a_j + \sum_j h_i w_{ij}) \tag{13}$$

The term $\langle v_i h_j \rangle_{model}$ can be obtained by performing alternative Gibbs sampling for a long time. The sampling starts from a random visible state. Then we update the hidden units in parallel with Equation 12, followed by updating the visible units in parallel with Equation 13. Instead of doing alternating Gibbs sampling for a large number of iterations, [15] proposed contrastive divergence (CD) as a faster learning procedure. The training also start with a training vector to compute the states of the hidden units using Equation 12. Then, with the chosen hidden states, we reconstruct the visible states by sampling each $v_i$ with probability given in Equation 13. The change of weight is then computed by

$$\Delta w_{ij} = \varepsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{reconstruct}) \tag{14}$$

This is called contrastive divergence using one full step of alternating Gibbs sampling. Contrastive divergence with $n$ rounds of alternating Gibbs sampling is usually denoted as CD$n$.

The layer-by-layer training algorithm for stacking RBMs goes in a greedy fashion. After learning the first layer RBM, the activity vector of the hidden units can be used as "data" for training the RBM in the second layer and this process can be repeated to learn as many hidden layers as desired. As data passing through the RBMs, we obtain the highest level features which are typically fed into a classifier. The entire deep network (RBMs plus the classifier) can be fine-tuned to improve the classification performance.

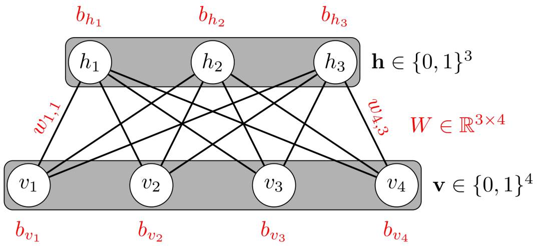

Figure 2: Restricted Boltzmann Machine. Figure courtesy of https://commons.wikimedia.org/wiki/File:Restricted-boltzmann-machine.svg

## 5.3 Autoencoders

An autoencoder neural network is an unsupervised model with typically one hidden layer that tries to set the output layer to be equal to the input. As shown in Figure 3, we want the network to learn a function $h_{W,b}(x) \approx x$. However, to prevent the network from learning the meaningless identity function, we need to place extra constraints on the network, giving birth to different flavors of autoencoders. In this project we consider two most popular types of autoencoder, sparse autoencoder and denoising autoencoder.

The **denoising autoencoder** algorithm is proposed by [38] and illustrated in Figure 4. To prevent learning identity function, an example $\mathbf{x}$ is first corrupted, either by adding Gaussian noise or by random masking a fraction of items in $\mathbf{x}$ to zero. The autoencoder then maps corrupted $\tilde{\mathbf{x}}$ to a hidden representation $\mathbf{y} = sigmoid(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b})$. From $\mathbf{y}$ we reconstruct $\mathbf{z} = g'_{\theta}(\mathbf{y})$. The training needs to learn the parameters $\theta$ and $\theta'$ so that average reconstruction error is minimized over training set. For binary input $\mathbf{x}$, usually cross entropy is adopted as $L_H(\mathbf{x}, \mathbf{z})$; while mean squared error is used for real-valued $\mathbf{x}$.

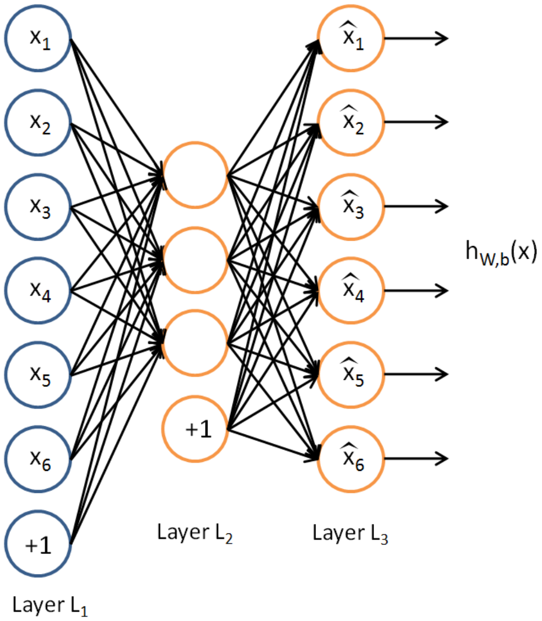

Figure 3: General Architecture of Autoencoders. Figure courtesy of [1].

The **sparse autoencoder** works by placing a sparsity constraint on the hidden units [31]. First, we make the autoencoder's hidden layer size to be over-complete, that is, of larger size comparing to the dimension of the input. Let's denote the activation of hidden unit $j$ of layer 2 in Figure 3 to be $a_j^2(\mathbf{x})$ given input example

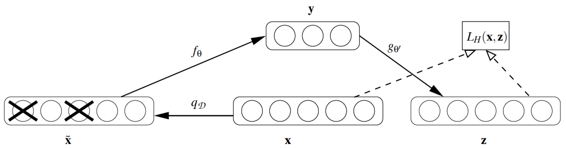

Figure 4: The denoising autoencoder algorithm. Input example $\mathbf{x}$ is randomly corrupted via $q_\mathcal{D}$ and then is mapped via encoder $f_\theta$ to $\mathbf{y}$. The decoder $g'_\theta$ attempts to reconstruct $\mathbf{x}$ and produces $\mathbf{z}$. Reconstruction error is measured by loss $L_H(\mathbf{x}, \mathbf{z})$, to be minimized during the training phase. Figure courtesy of [38].

$\mathbf{x}$. With that, we define the average activation of hidden unit $j$ over the $m$-size training set

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} a_j^2(\mathbf{x}) \tag{15}$$

The sparsity constraint is enforcing, ∀ hidden unit $j$,

$$\hat{\rho}_j = \rho \tag{16}$$

where $\rho$ is a sparsity parameter that approximates zero (say 0.05). This constraint can be vectorized over the hidden layer, say of size $n_2$, with the KL divergence based penalty term

$$\sum_{j}^{n_2} KL(\rho || \hat{\rho}_j) = \sum_{j}^{n_2} [\rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}] \tag{17}$$

The sparsity penalty term is integrated into the cost function by adding another hyper-parameter $\beta$

$$L(W, b) = \frac{1}{2} || h_{W,b}(\mathbf{x}) - \mathbf{x} ||^2 + \beta \sum_{j}^{n_2} KL(\rho || \hat{\rho}_j) \tag{18}$$

Denoising autoencoder and sparse autoencoder, surprisingly, have different application domains. Vincent et al. [38] have shown that stacked denoising autoencoder can be used to initialize a deep neural network's weight parameter, achieving similar and sometimes better performance than stacked RBM. They also show that training stacked denoising autoencoder with MNIST dataset, it is able to re-synthesize a variety of similarly good quality digits. Raina et al. [31] have compared sparse encoding with principle component analysis (PCA) and argue that transferring raw features with a well unsupervised trained sparse autoencoder can be beneficial to supervised learning algorithms, for example support vector machines (SVM).

### 5.4 Generative Adversarial Nets

As another generative model, Generative Adversarial Nets (GAN) [12] adopts a novel training framework, in which two models are trained simultaneously and adversarially. The generative model $G(z; \theta_g)$ aims to capture the probability distribution of the available unlabelled dataset, where its input is a noise variable $z$ following a prior distribution $p_z$. The discriminative model $D(x; \theta_d)$ output the probability distribution that whether the its input source $S$ comes from training dataset ($x \sim data$) or the generative model ($x \sim G(z)$):

$$D(X) = P(S|X) \tag{19}$$

Models $G$ and $D$ can be as simple as multilayer perceptrons, or as complex as deep convolutional nets when the task domain is image. The two models are trained in opposition to one another, with respect to the following log-likelihood function

$$\begin{aligned} V(D,G) &= \mathbb{E}_{\boldsymbol{x}\sim data}[\log P(S = real|X = \boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim G(\boldsymbol{z})}[\log P(S = fake|X = \boldsymbol{x})] && (20)\\ &= \mathbb{E}[\log D(\boldsymbol{x})] + \mathbb{E}[\log(1 - D(G(\boldsymbol{z})))] && (21)\end{aligned}$$

With $V(D,G)$ properly defined, the training procedure is a two-player minimax game. First we maximize the log-likelihood that $D$ correctly recognize both the training examples and the samples generated from $G$; in the following phase, we train $G$ to generate samples that trick $D$ to make most mistakes. This two-phase min-max optimization can be summarized as:

$$\min_{G}\max_{D} V(D,G) \tag{22}$$

Powerful though GAN is, large amount of efforts and care are needed during training. One way to make the training stable and fast is to augment GAN with an auxiliary classifier so that the training phase employs the labels available in the dataset [29]. In auxiliary classifier GAN (AC-GAN), the discriminator $D$ now gives both the probability distribution over the sources (whether $\boldsymbol{x}$ is real or fake) and the probability distribution over the class labels:

$$D(X) = P(S|X), P(C|X) \tag{23}$$

Accordingly, the log-likelihood function $V(D,G)$ is augmented with the log-likelihood of the correct class $L_C$:

$$\begin{aligned} V(D,G) &= L_S + L_C && (24)\\ L_S &= \mathbb{E}_{\boldsymbol{x}\sim data}[\log P(S = real|X = \boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim G(\boldsymbol{z})}[\log P(S = fake|X = \boldsymbol{x})] && (25)\\ L_C &= \mathbb{E}_{\boldsymbol{x}\sim data}[\log P(C = c|X = \boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim G(\boldsymbol{z})}[\log P(C = c|X = \boldsymbol{x})] && (26)\end{aligned}$$

The training procedure for AC-GAN is similar to GAN: we train $D$ to maximize $V(D,G)$; while at the same time we train $G$ to minimize $L_S - L_C$. Currently, we are interested in using GAN or AC-GAN to generate fake traffic. In the future, we will also attempt the semi-supervised classification framework with AC-GAN.

### 5.5 Wide and Deep Learning with Embeddings

In the network intrusion dataset, categorical and integer features are extremely sparse. Usually neural networks are not good at handle large sparse inputs. For illustration reason, we plot the histogram of the “dloss” integer feature in UNSW-NB15 dataset, which denotes the number of destination packets retransmitted or dropped. As shown in 5 the feature value ranges from 0 to 6000, while more than 97% of the occurred value is 0 but values from 1000 to 6000 do appear in the dataset. For categorical feature “proto”, it tells which one of the 133 protocol types the traffic record belongs to. If we one-hot encode “proto”, this feature will become a set of 133 features with only one being 1. If we convert “proto” to an feature of integer identifier, its value will range from 1 to 133.

Such specific situation can be tackled by two ways. The first solution is to embed the integer or categorical features. Simply put, an embedding is a mapping from sparse discrete objects to a dense vector of real numbers. It is widely used, also known as “word2vec”, in the natural language processing and machine translation tasks, where embeddings are treated as points in vector space such that similarity between objects can be visually measured by the Euclidean distance or angle between vectors.

In our case, embedding provide a solution to converting large vocabulary sized categorical features and sparse integer features to dense vectors of continuous values. Deep neural network fed with embedding inputs can generalize better even with less feature engineering. As stated in [9], these input features to the deep neural nets are denoted as deep components, consisted of continuous, one-hot encoded, and embedded features.

On the other hand, one can leverage simple linear models with nonlinear feature transformations to deal with sparse inputs, namely the wide components proposed in [9]. The wide components consist of two parts: the basis and crossed features. The basis features are raw input features that are either integer or categorical. The crossed features are cross-product transformations of basis features:

$$\Phi_k(\boldsymbol{x}) = \prod_{i=1}^{d} {x_i}^{c_{ki}} \tag{27}$$

where

$$c_{ki} = \begin{cases} 1, & i\text{-th feature} \in \text{the transformation } \Phi_k \\ 0, & \text{otherwise} \end{cases} \tag{28}$$

Since such wide components can memoize the feature interactions, it actually complements a deep neural network which generalizes better than linear model on the condition that inputs are low-dimension dense real value vectors. As shown in Figure 6, the wide and deep model combines the wide components and deep components using a weighted sum of both models' outputs.

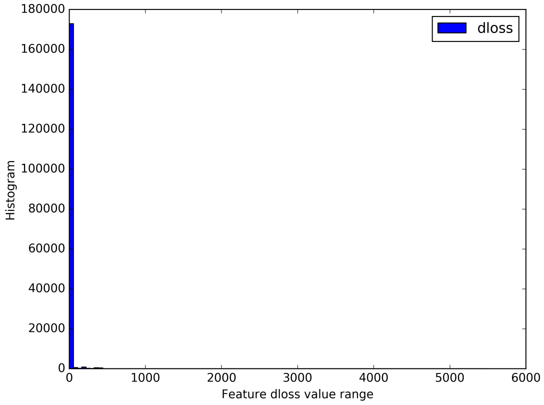

Figure 5: Histogram of the Feature "dloss" from UNSW-NB15 Dataset.

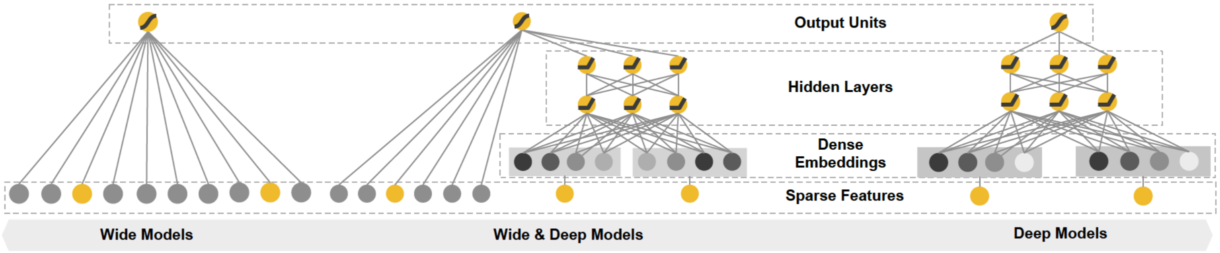

Figure 6: Anatomy of Wide and Deep Model

# 6 Implementation

## 6.1 TensorFlow 101

TensorFlow [5] is an open-source software library for machine learning developed by the Google Brain Team. The library models the computations in machine learning as data flow graphs. Multidimensional data arrays are called tensors in TensorFlow. Nodes in the graph represent mathematical operations between tensors, such as add, multiply, softmax and dropout. Graph edges represent the flow of tensors between nodes. The computation graph based architecture allow researchers to run or train neural networks on one or more CPUs or GPUs with unified API.

For general classification task, input $X$ and label $Y$ are defined as **placeholder**s and feed into the computation graph at running time using a dictionary. The graph of a typical deep learning model have three parts. The **inference** graph should be built so that output predictions are returned as tensor. For example, in the multilayer perceptron case, inference graph contains all iterative computation 4-5 in the feed-forwarding steps. The **loss** graph should compute the loss function defined by specific models or algorithms. Usually it is either cross-entropy or mean-squared error averaged across the batch data. The loss graph will be optimized, usually minimized, by the **train** part. This optimization can be conducted by various optimizing algorithms, such as gradient descent, Momentum, RMSProp. After sufficient steps of batch training, we evaluate the trained model with inference graph and compare the predictions with the test dataset labels. TensorFlow also provides various useful utilities for training models and running experiments. Using a Saver, we are able to checkpoint the training process so as to restoring the model for further training or evaluation. Users create Summary nodes to log the snapshot of interest variables, which can be automatically visualized by TensorBoard.

## 6.2 NetLearner

We provide a Python library NetLearner [3] that wraps up several deep learning models on the basis of TensorFlow. NetLearner modularizes multilayer perceptron, restricted Boltzmann machine, sparse autoencoder and masking-noise autoencoder, all of which are used to perform the 5-class classification on the NSL-KDD dataset.

## 6.3 Hacks and Tricks

For the multiple layer perceptron, we tried a 4-hidden-layer network with very wide size in each layer, several hundreds for each layer. The accuracy on training set is very exiting, usually more than 96%. However, its performance on test dataset is not satisfactory. Instead we found out that a single hidden layer with only sixteen neurons has good accuracy. It is trained with stochastic gradient descent (SGD) for 20 epochs and batch size 100. During the training, learning rate decays from 0.1 exponentially with the base of 0.32. We did not include regularization in the model, but did apply dropout of keep probability 0.8. We denote this approach as MLP and show its detailed results in the later section.

We build a RBM with 200-hidden units to perform unsupervised feature learning first on the dataset. The learned features are then fed into a simple softmax regression classifier. We trained the RBM using CD1 (contrastive divergence using one full step to get the negative data), with batch size 10 for 40 epochs. The learning rate is initialized at 0.01 and decay exponentially with the base of 0.64. We denote this combination of RBM and softmax regression as RBM in the later section.

We also implemented the self-taught learning architecture proposed in [19, 31], adopting sparse autoencoder as the unsupervised feature learner. The learned features will then be used for classification by a Softmax regression classifier. We contact the author of [19] so that we can reproduce their implementation with the same hyper-parameters. For example, the hidden layer size of the sparse autoencoder is 64; the sparsity value $\rho$ is 0.25. Different from [19], we found that using regularization in neither autoencoder nor softmax regression is helpful. So we didn't include regularization term in the both autoencoder and softmax cost function. The autoencoder is trained with SGD for 1000 epochs and batch size 5000. Different from MLP, we used Adam optimizer during the training. The learning rate starts at 0.01 and decay exponentially with base of 0.6. We denote this approach as SAE and report its performance in the later section.

As a variation to the sparse autoencoder based self-taught learning architecture, we explore what will the performance be if we replace sparse autoencoder with denosing autoencoder. We simply use dropout to emulate the masking noise and build masking noise autoencoder, in which input is randomly masked out with keep probability of 0.4. The size of the denoising autoencoder is 100. The autoencoder is trained with SGD for 1000 epochs and batch size 5000. We trained denoising autoencoder in the same way as we trained sparse autoencoder. The result of this approach is labeled as DAE in the later section.

One thing to notice is that we use the same seed to randomly initialized the weights and biases of the softmax regression classifier such that the learned features from RBM, sparse autoencoder and denoising autoencoder are comparable. For the same reason, all the softmax regression classifiers used by RBM, sparse autoencoder and denoising autoencoder are trained with Adam optimizer of batch size 100 for 100 epochs, with exponentially decay learning rate starting at 0.01, with dropout technique of keep probability 0.8.

# 7 Experiment Results

## 7.1 Dataset and Preprocessing

Among alternative available datasets [2, 25, 27], we choose NSL-KDD dataset [37] and UNSW-NB15 dataset [26] to evaluate the performance of various proposed neural networks in the network intrusion detection.

### 7.1.1 NSL-KDD Dataset

The NSL-KDD dataset originates from the KDDCup 99 dataset [2], which was used for the third International Knowledge Discovery and Data Mining Tool Competition. NSL-KDD dataset addresses two issues of the KDDCup 99 dataset. First, it eliminates the redundant records existing in KDDCup 99, which takes up 78% and 75% of the records in train and test set, respectively. Second, it samples the dataset such that the fraction of the record from a difficulty level is inversely proportional to its difficulty. Both enhancements make NSL-KDD dataset more suitable for evaluating intrusion detection systems.

The train dataset consists of 125,973 TCP connection records, while the test dataset consists of 22,544 ones. A record is defined by 41 features, including 9 basic features of individual TCP connections, 13 content features within a connection and 9 temporal features computed within a two-second time window, and 10 other features. Connections in the train dataset are labeled as either normal or one of the 24 attack types. There are additional 14 types of attacks in the test dataset, intentionally designed to test the classifier's ability to handle novel attacks. The task of the classifier is to identify whether a connection is normal or one of the 4 categories of attacks, namely denial of service (DoS), remote to local (R2L), user to root (U2R) and probing, also known as 5-class classification problem.

### 7.1.2 UNSW-NB15 Dataset

Similar to KDDCup 99 dataset, the UNSW-NB15 dataset is generated by simulating normal activities and attack behaviors in a testbed. The simulation is conducted in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) and 49 features in the dataset is extracted by a chain of software tools also developed by ACCS. The structure of the features is also similar to that of KDDCup 99: 5 flow features, 13 basic features, 8 content features, 9 time features and 12 other features. The size of the dataset is 257,673 in term of flow records, 175,341 of which are used for training set and the rest are for testing. There are nine types of attacks in the dataset. The only common type of attack between UNSW-NB15 and NSL-KDD is DoS. The new attacks in UNSW-NB15 are analysis, backdoor, exploits, fuzzers, generic, reconnaissance, shellcode, and worms. In this project, we consider the 2-class classification problem for UNSW-NB15 dataset: the task of the classifiers is to predict a given traffic is either normal or malicious.

### 7.1.3 Preprocessing

Our data preprocessing starts with map the symbolic fields to a unique integer identifier. For example, a data record from NSL-KDD dataset be one of the 5 types of traffic. Therefore, its label will be mapped to 0, if normal, or to a number from 1 to 4 representing one of the four attacks. A symbolic feature, like "protocol" in both dataset, will be mapped to integer from 1 to $n$ where $n$ is the number of possible unique values. We one-hot-encode only the label and features with small $n$. That is, a feature or a label of value $x$ will be converted to a $n$-dimensional binary vector with the $x$th dimension set to 1 and others set to zero. Then we shuffled the data, together with its labels, so that later in the stochastic gradient descent learning phase, batch data are already randomized. At last, we perform the min-max normalization so that data values are all in the range of [0, 1].

## 7.2 Evaluation Metrics

We evaluate the classification performance of our proposed deep learning approaches with the following metrics.

- **Accuracy** is the percentage of correctly classified connections over the total number of connections in the dataset:

$$A = \frac{\text{Correct Predictions}}{\text{Number of Records}} \tag{29}$$

  Accuracy is not suitable for evaluating biased dataset where the number of records of some class is extremely larger than the number of records of another class. In NSL-KDD dataset, the number of available U2R records (67) is in two degrees of magnitude less than the other classes of traffic (9711, 7458, 2887, 2121 respectively). Therefore we also consider the following metrics.

- **Precision** is the percentage of the correctly classified positives over the total number of positives predicted by the classifier:

$$P = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{30}$$

- **Recall** is the percentage of the correctly classified positives over the total number of relevant elements:

$$R = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{31}$$

- **F1-Score** represents a balance between precision and recall and is calculated as their harmonic mean:

$$F = \frac{2PR}{P + R} \tag{32}$$

In the 5-class classification, we calculate the precision, recall and F1-Score for each traffic class. Additionally, we report the weighted average of these metrics as a single value for comparing various approaches. The weight for each class is determined by its proportion in the test dataset. The weight vector for class [Normal, Probe, DoS, U2R, R2L] is [0.431, 0.107, 0.339, 0.018, 0.105]. Besides, we also provide the confusion matrix of the classification results when applying different approaches on the test dataset. In our confusion matrix table, the row represents the instance in an actual class, while the column represents the instance in a predicted class. It is called confusion matrix because it is useful for visualizing how a classifier is confusing one class with other classes.

## 7.3 Performance of Deep Learning Approaches on NSL-KDD Dataset

First we report the classification accuracy of each considered approach in Figure 7. Surprisingly, the most "accurate" approach is the simple 16-neuron perceptron (81.42%). Sparse autoencoder based self-taught leaner achieved second best accuracy of 79.15%. This number coincides with the previously reported results in [19] (79.10%). RBM and denoising autoencoder have similar accuracy results (77.58% and 76.93%).

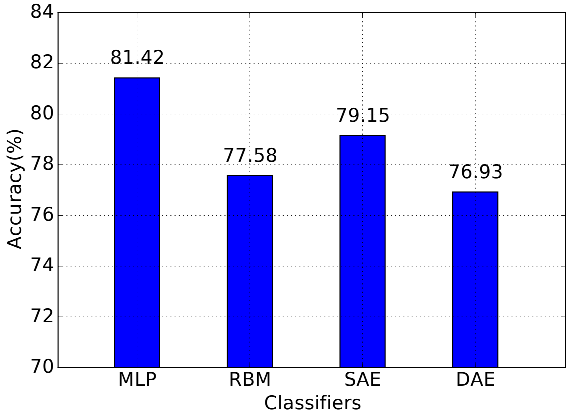

Figure 7: Classification Accuracy of Proposed Approaches on NSL-KDD Dataset

Table 2 – 5 summarize the confusion matrices of each approach and their weighted average metrics (Precision, Recall and F1-Score). Apart from best accuracy, MLP also achieved the best F1-Score (80.44%) among all of the considered approaches. However, we can see that for U2R attacks, MLP still has very poor results of both precision (13.95%) and recall (10.35%). The second highest F1-Score is achieved by sparse autoencoder combined with softmax regression (78.05%). This value is actually a little bit higher than the result (75.76%) reported in [19]. We believe this is partly due to the reason that we did not introduce regularization for both sparse autoencoder and softmax regression classifier, and partly due to the dropout technique we used in training the softmax regression classifier. RBM and DAE again achieve very similar classification performance, with F1-Score of 75.63% and 75.65% respectively. Considering both the high accuracy and best F1-Score, we conclude that in the competition of 5-class classification, MLP is the winner.

Confusion matrices here tell us something interesting about the classification performance for each type of traffics. MLP correctly recognized the most number of normal traffics (9329 out of 9711). For the attacking traffics, the winner classifier MLP correctly predicted the most number of DoS attacks (6146 out of 7636), U2R attacks (41 out of 396) and R2L attacks (926 out of 2376). RBM is the best classifier in predicting Probe attacks (2015 out of 2425).

Table 2: Confusion Matrix of MLP on Test Dataset

| | | Prediction | | | | |
|---|---|---|---|---|---|---|
| | | Normal | Probe | DoS | U2R | R2L |
| Actual | Normal | 9329 | 230 | 70 | 25 | 57 |
| | Probe | 164 | 1914 | 271 | 10 | 66 |
| | DoS | 1358 | 82 | 6146 | 47 | 3 |
| | U2R | 345 | 4 | 0 | 41 | 6 |
| | R2L | 1247 | 30 | 2 | 171 | 926 |
| Precision(%) | | 74.97 | 84.69 | 94.71 | 13.95 | 87.52 |
| Wtd. Avg.(%) | | | | | | 82.96 |
| Recall(%) | | 96.07 | 78.93 | 80.49 | 10.35 | 38.97 |
| Wtd. Avg.(%) | | | | | | 81.42 |
| F1-Score(%) | | 84.22 | 81.71 | 87.02 | 11.88 | 53.93 |
| Wtd. Avg.(%) | | | | | | **80.44** |

Table 3: Confusion Matrix of RBM on Test Dataset

| | | Prediction | | | | |
|---|---|---|---|---|---|---|
| | | Normal | Probe | DoS | U2R | R2L |
| Actual | Normal | 8903 | 318 | 428 | 11 | 51 |
| | Probe | 232 | 2015 | 159 | 2 | 17 |
| | DoS | 1879 | 143 | 5613 | 0 | 1 |
| | U2R | 356 | 3 | 1 | 27 | 9 |
| | R2L | 1550 | 8 | 1 | 8 | 809 |
| Precision(%) | | 68.91 | 81.02 | 90.50 | 56.25 | 91.21 |
| Wtd. Avg.(%) | | | | | | 79.65 |
| Recall(%) | | 91.68 | 83.09 | 73.51 | 6.82 | 34.05 |
| Wtd. Avg.(%) | | | | | | 77.04 |
| F1-Score(%) | | 78.68 | 82.04 | 81.12 | 12.16 | 49.59 |
| Wtd. Avg.(%) | | | | | | 75.63 |

## 7.4 Performance of Deep Learning Approaches on UNSW Dataset

We plot the prediction accuracies of different approaches in Figure 9. Support vector machine (SVM) is adopted as traditional machine learning approach that multiple neural networks compare to. We have trained both linear SVM and non-linear SVM with radial basis function kernel. Here we report the non-linear one's accuracy on test dataset because it is superior to linear one (81.50% v.s. 82.25%). We have also trained a plain neural network of a three layers with layer sizes of [480, 512, 640], denoted as Baseline Neural Network (BL-NN) in the following text. We use embedding to convert the sparse features in UNSW dataset, listed in Table 6. Its accuracy, more than 5% increase to SVM, also serves as baseline for the following two novel models described in Section 5.

The first model is auxiliary-classifier generative adversarial net (AC-GAN). Our training contains two phases. In the first stage, we trained an AC-GAN to generate fake normal and fake attacking traffics that equals the amount of normal and attacking traffics in training set respectively. In Figure 8, we plot 100 synthesized data for both normal and attacking traffic. Since the one-hot-encoded and normalized UNSW

Table 4: Confusion Matrix of SAE on Test Dataset

| | | Prediction | | | | |
|---|---|---|---|---|---|---|
| | | Normal | Probe | DoS | U2R | R2L |
| | Normal | 8864 | 696 | 92 | 11 | 48 |
| | Probe | 179 | 2001 | 164 | 2 | 79 |
| Actual | DoS | 1542 | 39 | 6054 | 0 | 1 |
| | U2R | 357 | 1 | 1 | 30 | 7 |
| | R2L | 1444 | 6 | 5 | 26 | 895 |
| Precision(%) | | 71.56 | 72.95 | 95.85 | 43.48 | 86.89 |
| Wtd. Avg.(%) | | | | | | 81.06 |
| Recall(%) | | 91.28 | 82.52 | 79.28 | 7.58 | 37.67 |
| Wtd. Avg.(%) | | | | | | 79.15 |
| F1-Score(%) | | 80.23 | 77.44 | 86.78 | 12.90 | 52.55 |
| Wtd. Avg.(%) | | | | | | 78.05 |

Table 5: Confusion Matrix of DAE on Test Dataset

| | | Prediction | | | | |
|---|---|---|---|---|---|---|
| | | Normal | Probe | DoS | U2R | R2L |
| | Normal | 9249 | 319 | 85 | 10 | 48 |
| | Probe | 576 | 1504 | 226 | 2 | 117 |
| Actual | DoS | 1842 | 128 | 5665 | 0 | 1 |
| | U2R | 353 | 1 | 0 | 38 | 4 |
| | R2L | 1469 | 3 | 1 | 17 | 886 |
| Precision(%) | | 68.57 | 76.93 | 94.78 | 56.72 | 83.90 |
| Wtd. Avg.(%) | | | | | | 79.75 |
| Recall(%) | | 95.24 | 62.02 | 74.19 | 9.60 | 37.29 |
| Wtd. Avg.(%) | | | | | | 76.93 |
| F1-Score(%) | | 79.73 | 68.68 | 83.23 | 16.41 | 51.63 |
| Wtd. Avg.(%) | | | | | | 75.65 |

dataset happens to have 121 features whose values all in the range of

$$0, 1$$

, traffic record generated by our AC-GAN is visualized as a $11\times 11$ gray images. We can see subtle difference between the normal and attacking traffic. Then we train a 400-neuron single-hidden-layer perceptron using both the synthesized and authentic data as training data. Unfortunately, comparing to the baseline three-layer neural network, AC-GAN does not provide significant improvement in terms of accuracy.

The second model we attempted is wide and deep learning model. As stated in Section 5.5, the wide and deep model (W&D) requires us engineering the features of UNSW dataset into basis, crossed, continuous, indicator and embedded features. The basis features are all the raw symbolic and integer features. These basis features are also fed to the deep model after conduct the embedding. The continuous features are all the raw continuous features. We make the full combinations of symbolic features to be crossed features. For comparison reason, we set the structure of the deep neural network in W&D to be the same sizes as that of the baseline neural network, namely three hidden layers with sizes of [480, 512, 640]. As the result shows, augmenting the deep neural network with wide linear regressor provides a 3% increase in accuracy to the

Table 6: Features Embeddings for the Three-Layer Perceptron

| Featue Name | Vocabulary Size | Embedding Size |
|---|---:|---:|
| state | 11 | 4 |
| service | 13 | 4 |
| protocol | 133 | 8 |

(a) 100 Normal Traffic Records

(b) 100 Attacking Traffic Records

Figure 8: Normal and Attacking Traffic Records Synthesized by AC-GAN

baseline three-layer neural network.

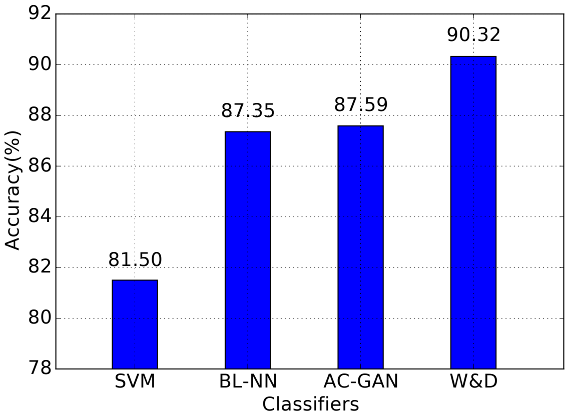

Figure 9: Classification Accuracy of Proposed Approaches on UNSW-NB15 Dataset

# 8 Conclusion

In this project we conducted a comparative study on the deep learning approaches for the network intrusion detection problem. We take the off-line network intrusion detection dataset NSL-KDD for evaluation. In this paper, multilayer perceptron, restricted Boltzmann machine, sparse autoencoder and denoising autoencoder are briefly described. The main contribution lies on sharing the hacks and tricks used in training these neural networks and the results of comparable evaluation of them. From our experiment results, we conclude that for the NSL-KDD test dataset, multilayer perceptron has relatively best performance since both its accuracy and F1-Score are outstanding among compared neural networks.

# References

[1] Autoencoders. `http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/`. Accessed: 2017-3-3.

[2] KDD Cup 1999 Data. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`. Accessed: 2017-3-10.

[3] NetLearner. `https://github.com/littlepretty/NetLearner`. Accessed: 2017-7-14.

[4] SNORT. `https://www.snort.org/`. Accessed: 2017-4-15.

[5] TensorFlow: An open-source software library for Machine Intelligence. `https://www.tensorflow.org/`. Accessed: 2017-4-15.

[6] C. Angulo, X. Parra, and A. Catal. K-svcr. a support vector machine for multi-class classification. *Neurocomputing*, 55(1):57 – 77, 2003. Support Vector Machines.

[7] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 93–104, New York, NY, USA, 2000. ACM.

[8] P. Casas, J. Mazel, and P. Owezarski. Unsupervised network intrusion detection systems: Detecting the unknown without knowledge. *Computer Communications*, 35(7):772 – 783, 2012.

[9] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide and Deep Learning for Recommender Systems. *ArXiv e-prints*, June 2016.

[10] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[11] U. Fiore, F. Palmieri, A. Castiglione, and A. D. Santis. Network anomaly detection with the restricted boltzmann machine. *Neurocomputing*, 122:13–23, 2013.

[12] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *ArXiv e-prints*, Dec. 2015.

[14] G. Hinton. A practical guide to training restricted boltzmann machines. Technical Report UTML TR-2010-003, Department of Computer Science, University of Toront, 6 King's College Rd, Toronto, August 2010.

[15] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800, Aug. 2002.

[16] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[17] S. M. Hosseini Bamakan, H. Wang, and Y. Shi. Ramp loss k-support vector classification-regression; a robust and sparse multi-class approach to the intrusion detection problem. *Know.-Based Syst.*, 126(C):113–126, June 2017.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[19] A. Javaid, Q. Niyaz, W. Sun, and M. Alam. A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies, New York, NY, USA*, volume 35, pages 2126–2132, 2015.

[20] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*, Dec. 2014.

[21] R. Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 202–207. AAAI Press, 1996.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[23] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. *A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection*, pages 25–36.

[24] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[25] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. The 1999 DARPA Off-line Intrusion Detection Evaluation. *Computer Networks*, 34(4):579–595, Oct. 2000.

[26] N. Moustafa and J. Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6, Nov 2015.

[27] N. Moustafa and J. Slay. The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. *Inf. Sec. J.: A Global Perspective*, 25(1-3):18–31, Apr. 2016.

[28] N. Moustafa, J. Slay, and G. Creech. Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks. *IEEE Transactions on Big Data*, PP(99):1–1, June 2017.

[29] A. Odena, C. Olah, and J. Shlens. Conditional Image Synthesis With Auxiliary Classifier GANs. *ArXiv e-prints*, Oct. 2016.

[30] J. R. Quinlan. *Learning Efficient Classification Procedures and Their Application to Chess End Games*, pages 463–482. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

[31] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng. Self-taught learning: Transfer learning from unlabeled data. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 759–766, New York, NY, USA, 2007. ACM.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of Research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[35] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147. JMLR.org, 2013.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. *ArXiv e-prints*, Sept. 2014.

[37] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6, July 2009.

[38] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.