IBM Security

Neural Log Classifier

A TOP - DOWN APPROACH: FROM AI TO NLP

Niccolò Howard Minetti
Thesis Student

– 03 – 19

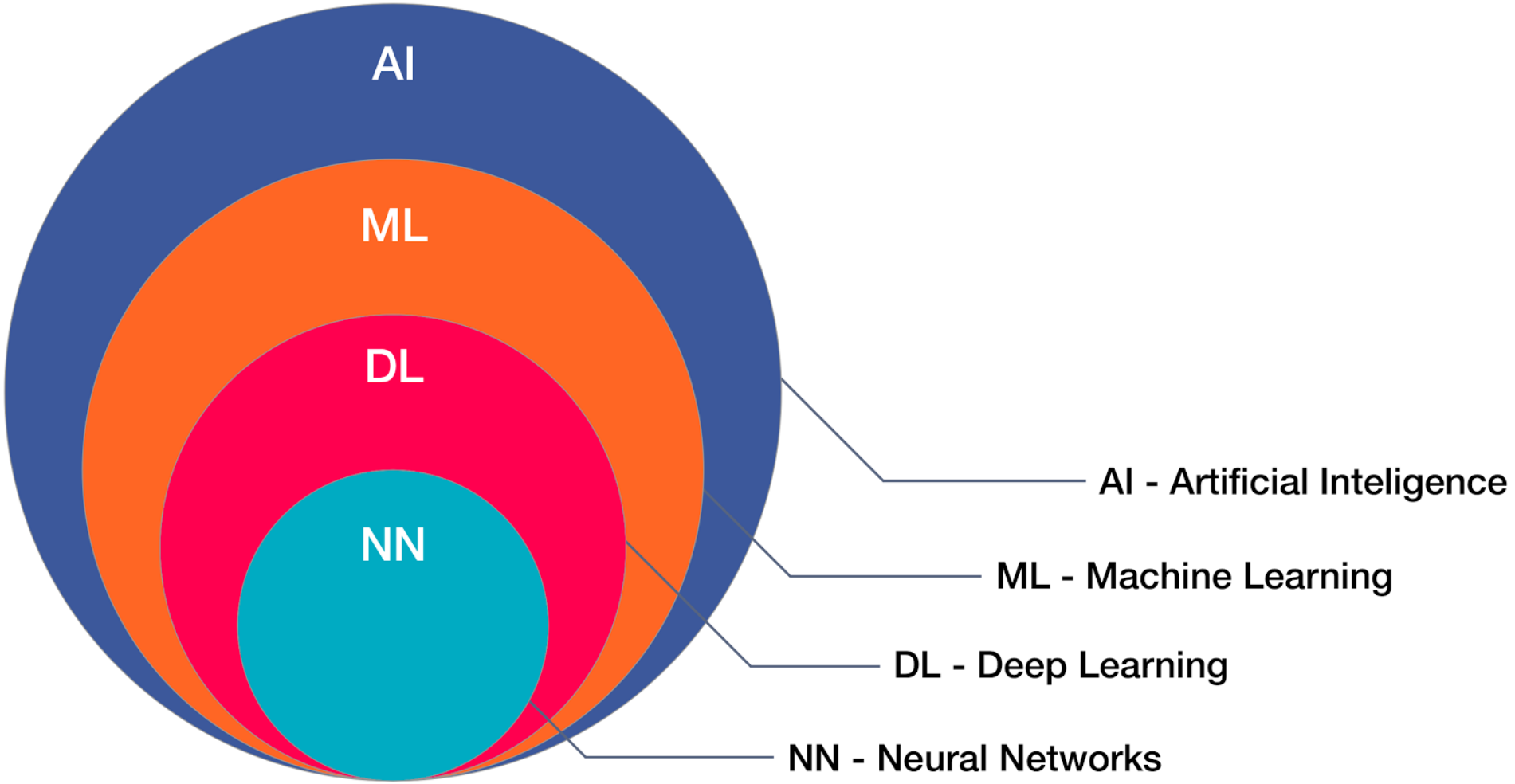
IBM

Agenda

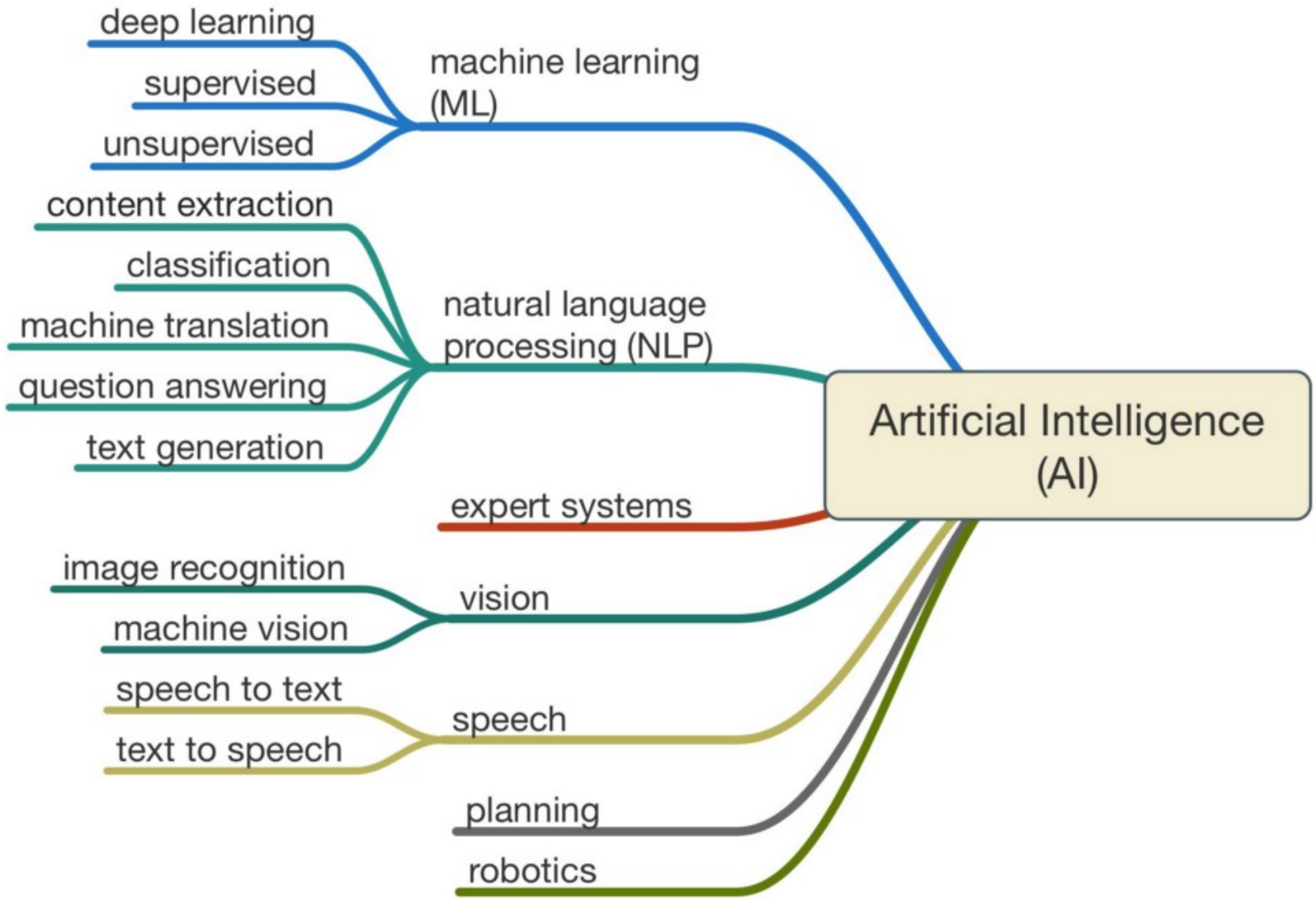
- Introduction to AI
  - Ai, ML and DL
  - Pillars for reference
- Natural Language Processing
  - Pipeline Concepts
- Neural Log Classifier Project
  - Objective
  - Proof of concept architecture
  - Future Development

Introduction to AI

AI, ML and DL

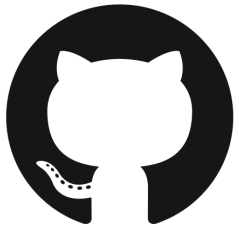

AI
ML
DL
NN
AI - Artificial Inteligence
ML - Machine Learning
DL - Deep Learning
NN - Neural Networks

AI, ML and DL


Artificial Intelligence (AI)
machine learning (ML)
deep learning
supervised
unsupervised
natural language processing (NLP)
content extraction
classification
machine translation
question answering
text generation
expert systems
vision
image recognition
machine vision
speech
speech to text
text to speech
planning
robotics

Pillars of Reference

OpenAI



MIT Technology Review

Recent news – 2 Weeks (March 2019)

OpenAI

- **Better Language Models and Their Implications**
  - Given a certain statement, this novel algorithm can create indistinguishable fake articles motivating such statement

An AI that writes convincing prose risks mass-producing fake news

Fed with billions of words, this algorithm creates convincing articles and shows how AI could be used to fool people on a mass scale.

by Will Knight February 14, 2019

Google AI

- **Using Global Localization to Improve Navigation**
  - Visual anchors are used in most AR/VR for orientation

Google is letting some users test its AR navigation feature for Google Maps

The feature will 'come to everyone only when Google is satisfied that it's ready'

By Andrew Liptak | @AndrewLiptak | Feb 10, 2019, 3:11pm EST

SHARE

DeepMind

- **Reinforcement learning goes real-time**
  - Gamification is one of the best ways of quantifying sucess

AlphaStar: Mastering the Real-Time Strategy Game StarCraft II

Games have been used for decades as an important way to test and evaluate the performance of artificial intelligence systems. As capabilities have increased, the research community has sought games with increasing complexity that capture different elements of intelligence required to solve scientific and real-world problems. In recent years, StarCraft, considered to be one of the most challenging Real-Time Strategy (RTS) games and one of the longest-played esports of all time, has emerged by consensus as a "grand challenge" for AI research.

IBM

Alright, so how did we go from data science to artificial intelligence anyway? The more I try to define “data science” the more I just… I just… I can’t describe it. The whole thing is just so insane and vague.”

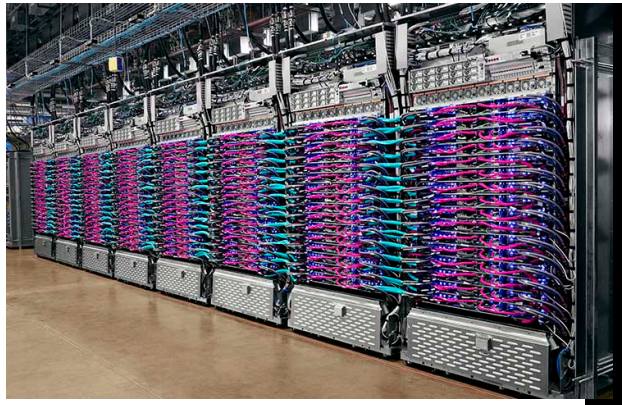
THOMAS NIELD

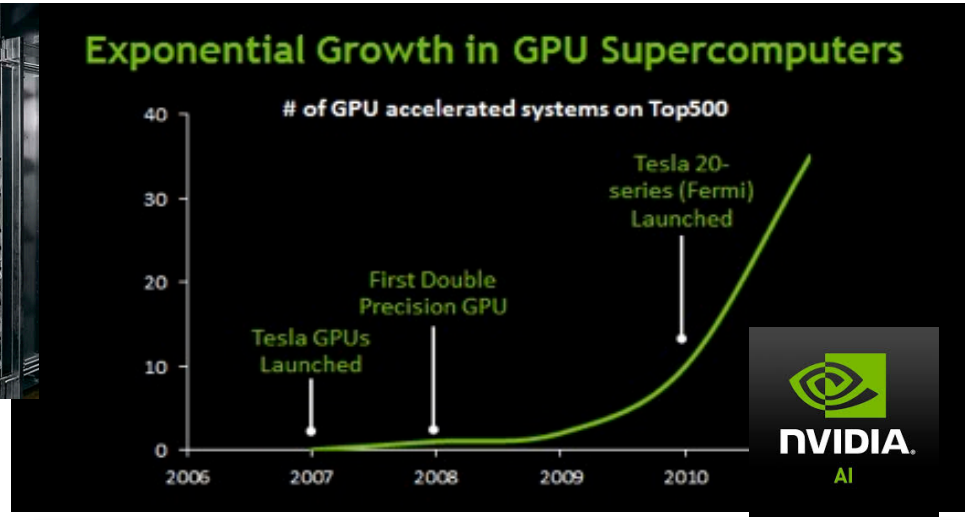
How It Feels to Learn Data Science in 2019

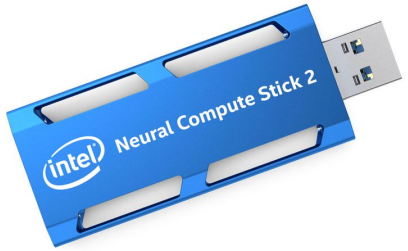
Medium

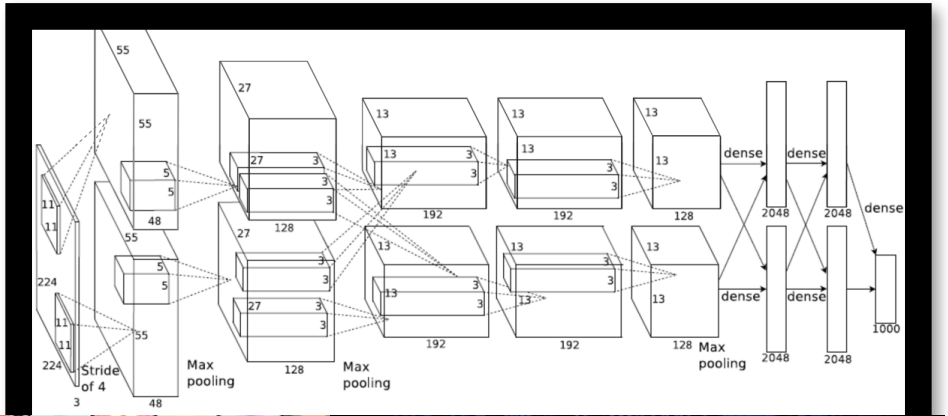
Why now?

- GPU Power (+TPU)
- Algorithms
- From Big Data to Data Sets





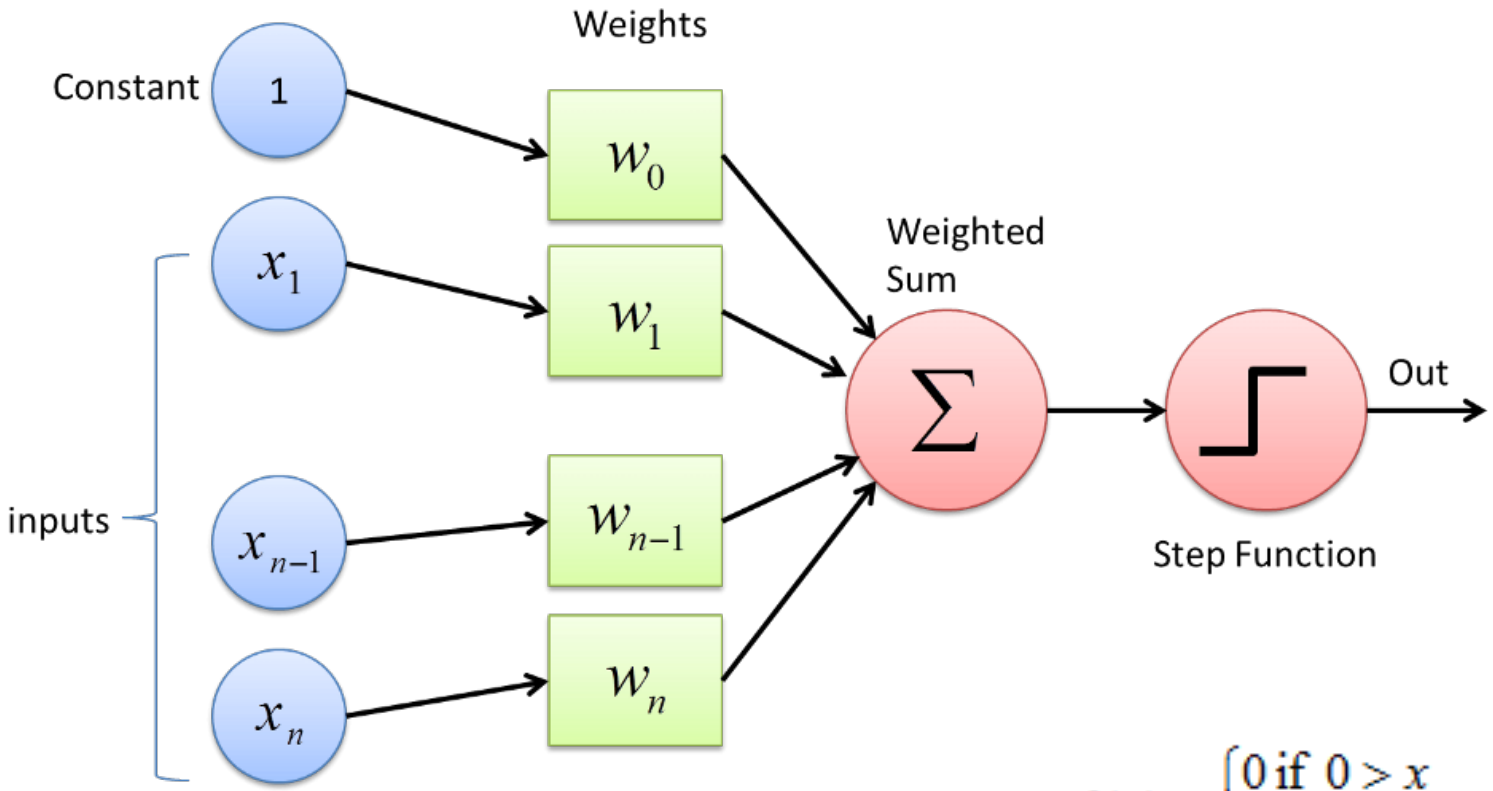




kaggle IMAGENET

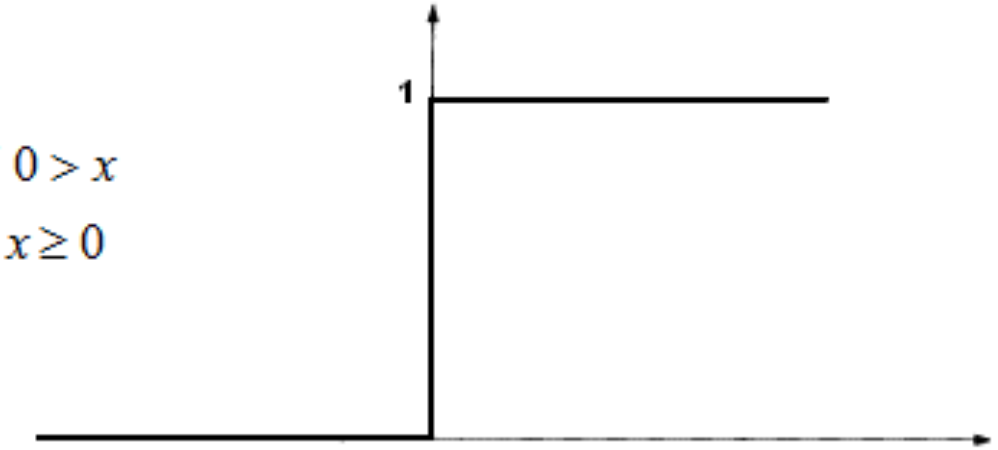
Architecture overview

Perceptron = Neuron

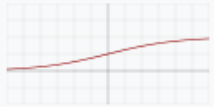
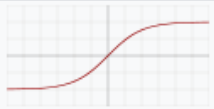


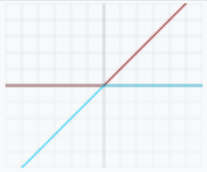
Unit step (threshold)

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



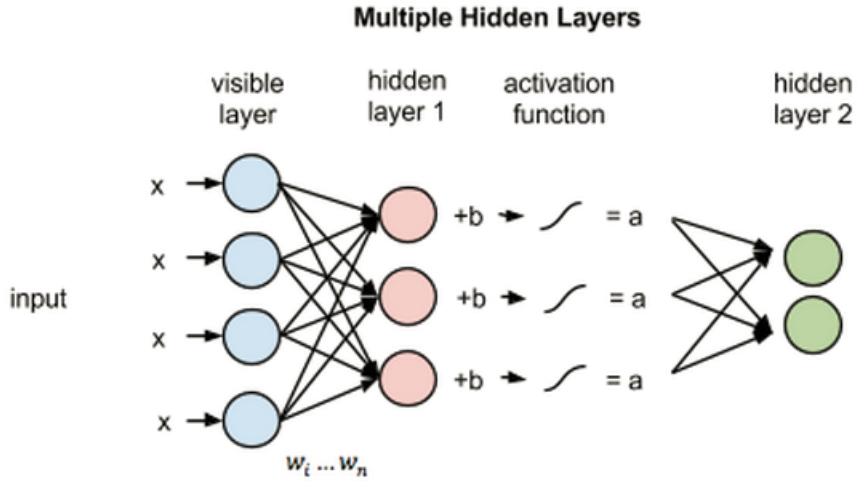
Activation Functions

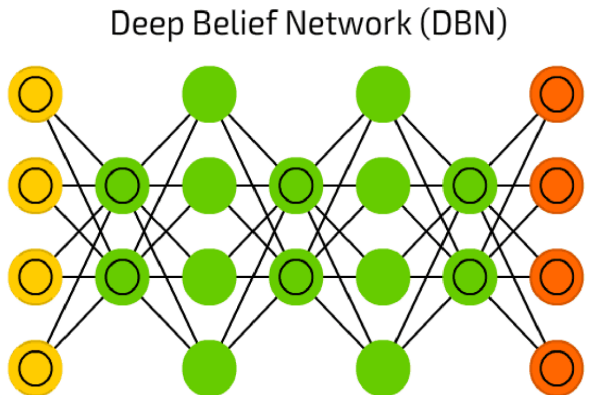
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$ [1]	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$

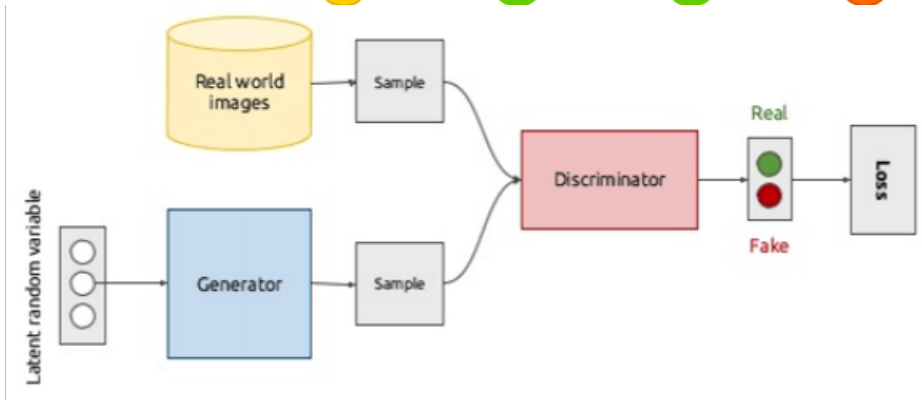
Rectified linear unit (ReLU)[15]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Bipolar rectified linear unit (BReLU)[16]		$f(x_i) = \begin{cases} ReLU(x_i) & \text{if } i \bmod 2 = 0 \\ -ReLU(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$f'(x_i) = \begin{cases} ReLU'(x_i) & \text{if } i \bmod 2 = 0 \\ -ReLU'(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$(-\infty, \infty)$
Leaky rectified linear unit (Leaky ReLU)[17]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Parameteric rectified linear unit (PReLU)[18]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$[2]
Randomized leaky rectified linear unit (RReLU)[19]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$[3]	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$

Unsupervised Learning

- Autoencoder: Restricted Boltzmann Machine (RBM)
  - Applies a dimensionality reduction transformation
  - Tries to rebuild the input from the reduction
  - Result: finds and refines pattern features
- Deep Belief Networks (DBN)
  - Stack of RBMs
  - Why stack layers? Different dimensionality features
- Generative Adversarial Networks (GANs)
  - Two networks, a classifier and a generator fight over who is right

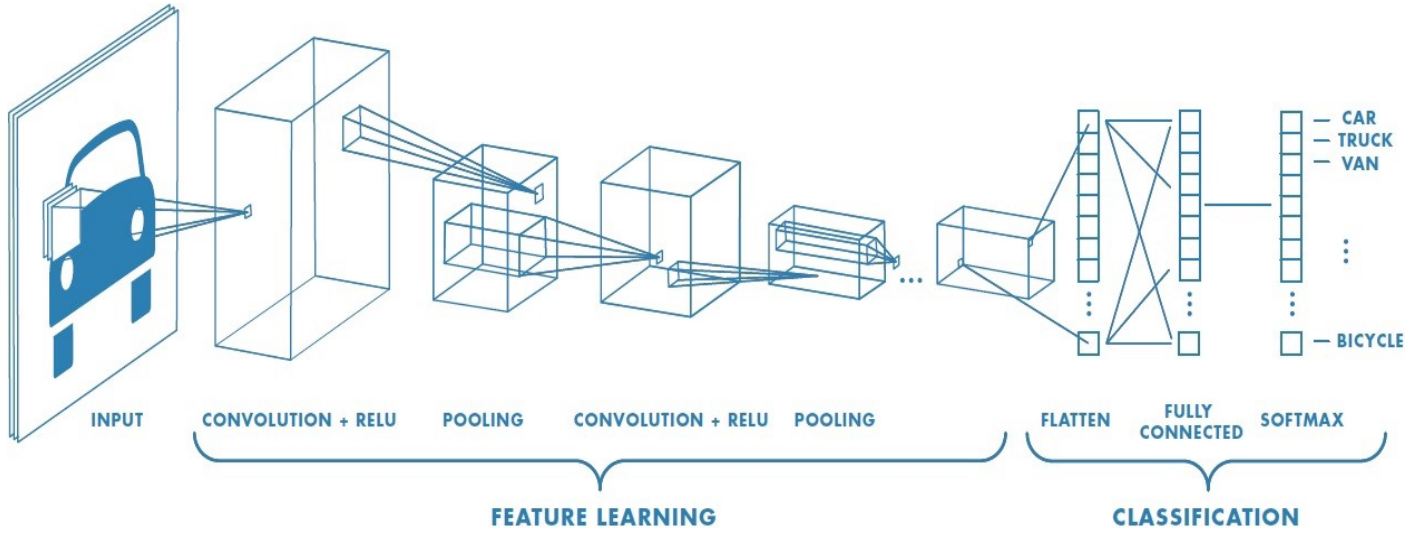




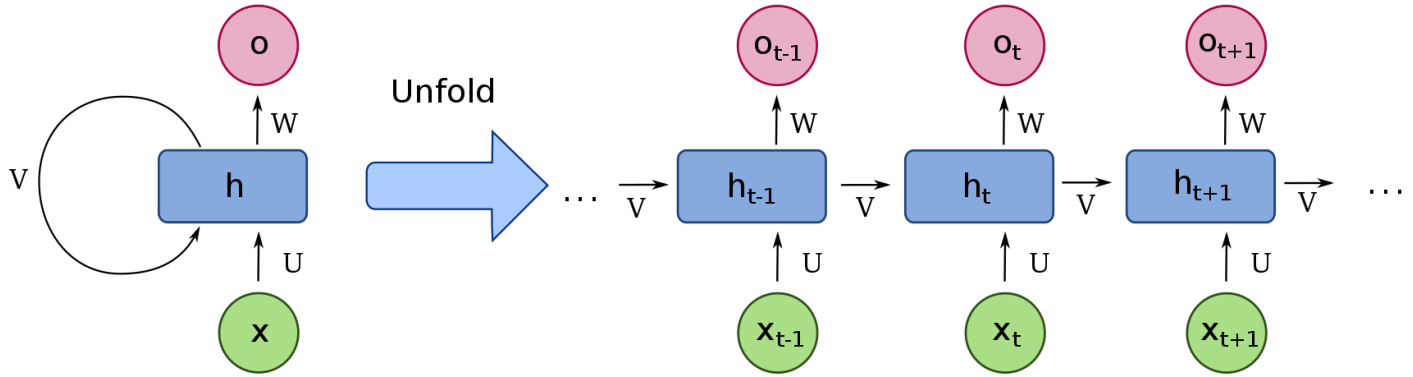


Supervised Learning

- Convolutional Neural Networks (CNN)
  - A series of different layers
  - Different activation functions
  - Very good for image recognition

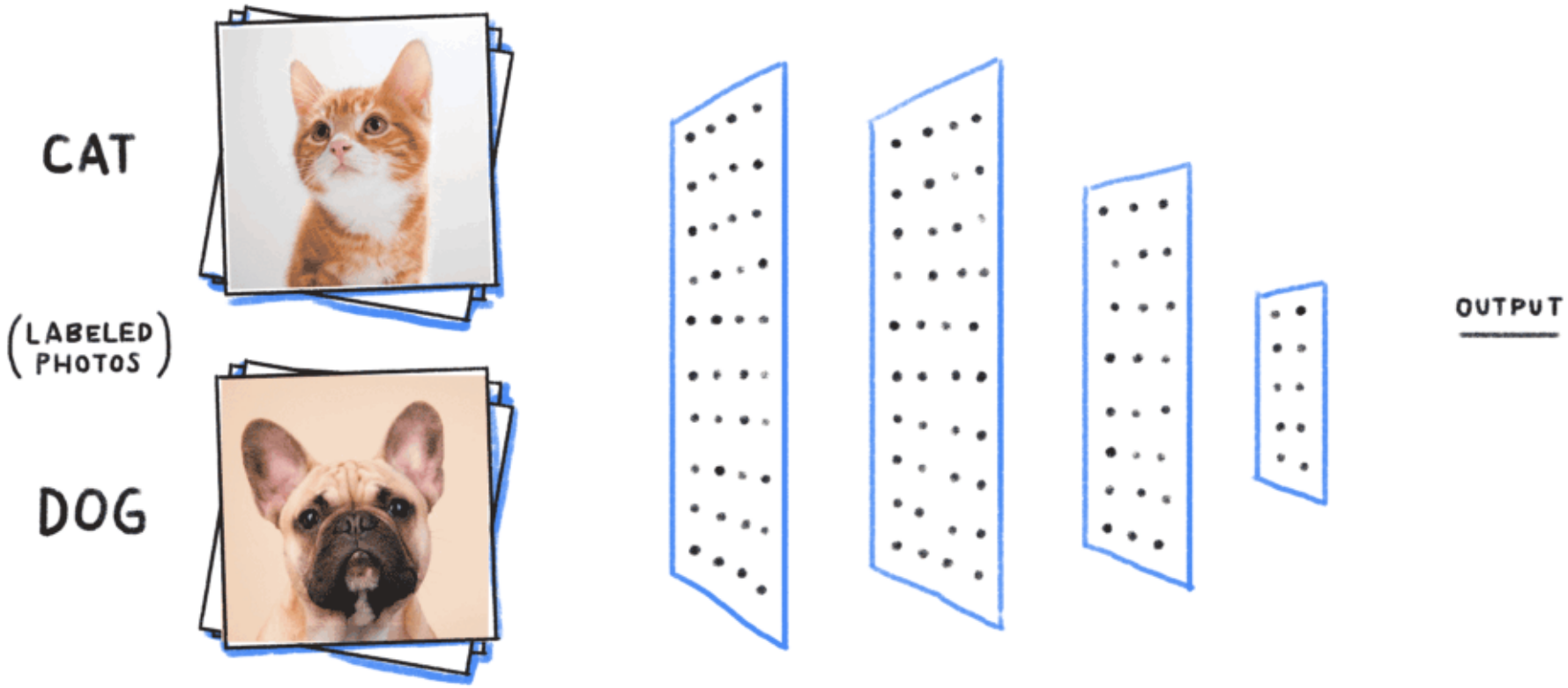


- Recurrent Neural Networks (RNNs)
  - For time-dependant features
  - Self-connected



CNN

Many Perceptrons = Neural Network



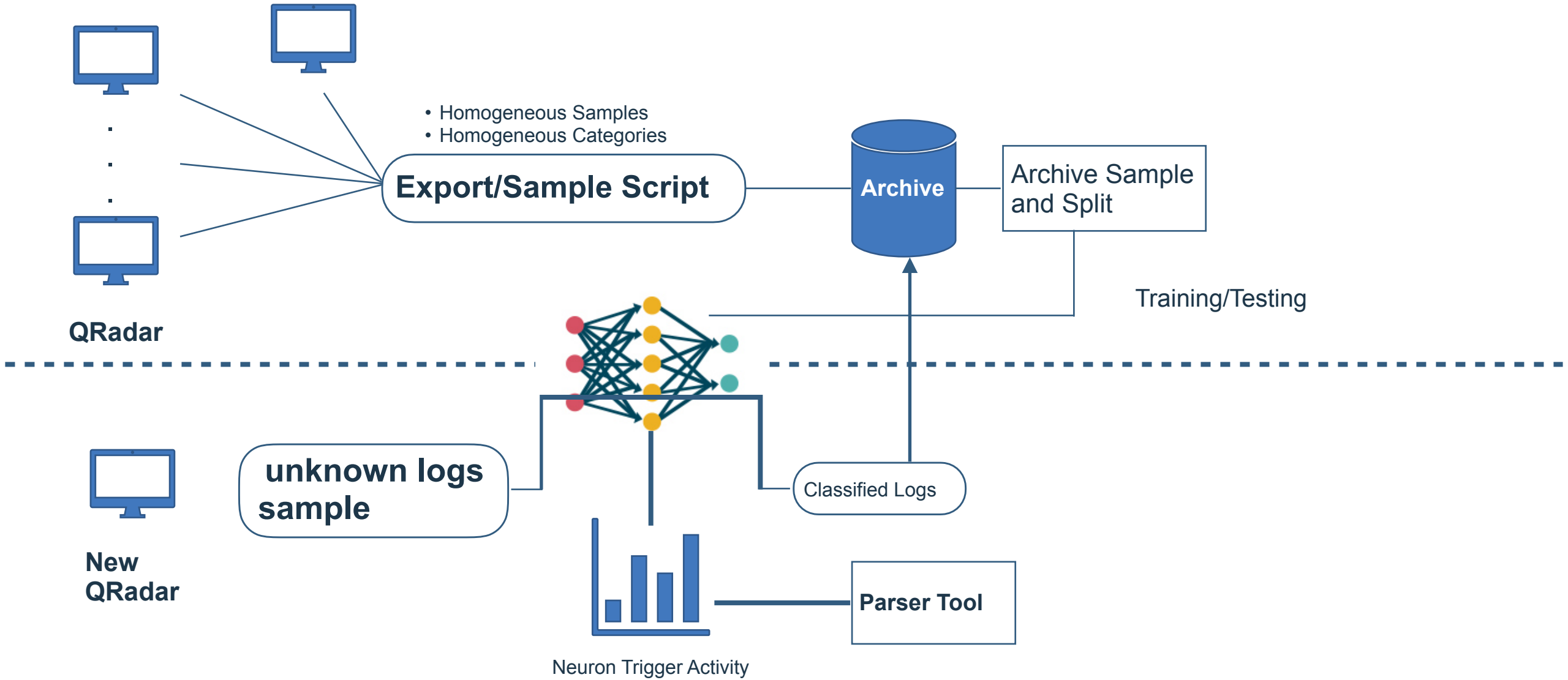
Natural Language Processing

Natural language processing (**NLP**) is a subfield of computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyse large amounts of natural language data.

WIKIPEDIA
Natural Language Processing

Neural Log Classifier

Architecture



Objective

- **What:** Reduce deployment time of QRadar.

- **How:** The objective of the POC is to build a lightweight neural network that takes in as input raw “utf_payload” and suggests in which category such Log should be places along with the features that where triggered in making this decision.

- **Methods:** Natural Language Processing (NLP) using Keras, Tensorflow and Python

POC Architecture

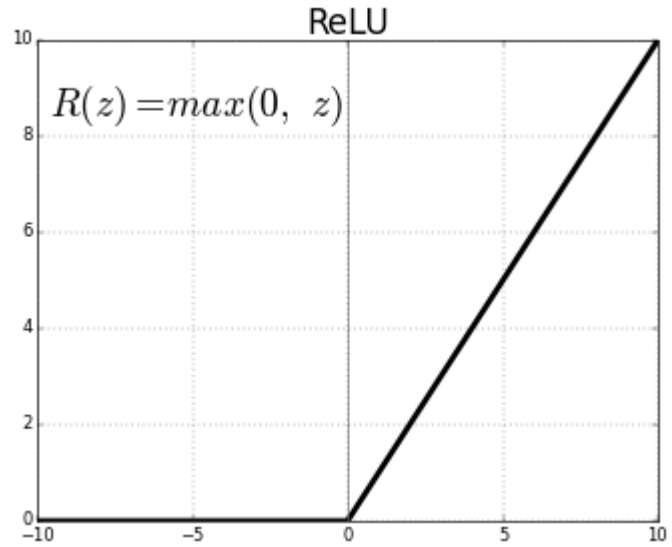
- Cleaning/Sampling:
  - Log activity is usually dishomogeneous, mainly dominated by firewall permits
  - We need equality distributed categories
  - Datadump of Logs where we extract a well distributed dataset (only UTF payloads and categories)

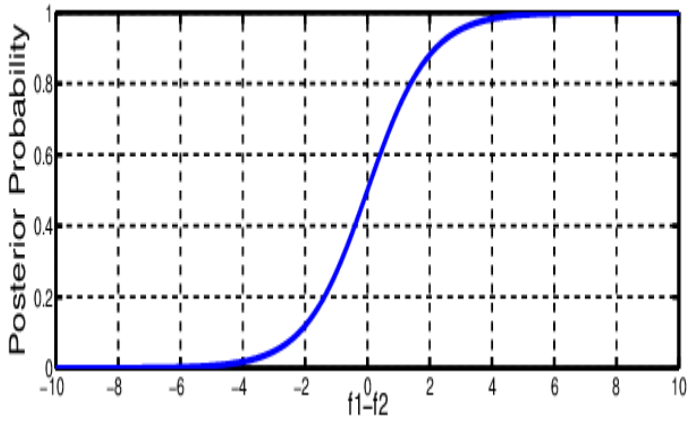
- Tokenization:
  - We need to translate words into numbers
  - We scan for most used 500 words in all logs to create a dictionary to map element occurences
  - Each log is then Tokenized (mapped) with the dictionary we have just built

- Neural Net:
  - Classifier that takes as input a Tokenized log and gives as output suggested categories with probability

Architecture

- Sequential Model: Stack of Layers
- Input: Log encoding
- First Layer:
  - Input: Vectorized Log
  - Activation function: ReLu
  - Output size: 512
- Second Layer:
  - Input: ReLu output
  - Activation function: Softmax
  - Output: number of categories found
- Output: Log Category





Code

```
label = 'categoryname_category' #Insert 'categoryname_category' or 'categoryname_highlevelcategory'

data = pd.read_pickle("NLP_5k+2")          #Load homogeneous sample of 5K logs
data = data.sample(frac=1).reset_index(drop=True)  #Shuffle elements


train_size = int(len(data) * .8)                    #80-20 division for training-testing
train_rLogs = data['utf8_payload'][:train_size]
train_cat = data[label][:train_size]
test_rLogs = data['utf8_payload'][train_size:]
test_cat = data[label][train_size:]

labelNum = len(train_cat.value_counts())            #Counting the number of labels in the subset
print("Found:", labelNum, "Categories \n")

vocab_size = 500                                    #Size of dictionary
tokenize = text.Tokenizer(num_words=vocab_size)     #init Tokenizer
tokenize.fit_on_texts(train_rLogs)                  #Fit utf_payload to dictionary

x_train = tokenize.texts_to_matrix(train_rLogs)     #Maps logs to matrix
x_test = tokenize.texts_to_matrix(test_rLogs)

encoder = LabelBinarizer()
encoder.fit(train_cat)                              #Fits labels for training

y_train = encoder.transform(train_cat)
y_test = encoder.transform(test_cat)
```

Code

```
#Linear stack of layers

model = []
model = Sequential()

#First Layer

model.add(Dense(512, input_shape=(vocab_size,)))     #Dense layers applies y =ax+b with a Activation function
                                                     #with 512 neurons and an input the size of our dictionary

model.add(Activation('relu'))                        #ReLu: Rectified Linear Unit (Activation function)

#Second Layer

model.add(Dense(labelNum))


model.add(Activation('softmax'))                     #Softmax activation function

model.compile(loss='categorical_crossentropy',       #Categorical CrossEntropy is CrossEntropy with Softmax
              optimizer='adam',                      #Adaptive moment estimation for gradient optimisation
              metrics=['accuracy'])

history = model.fit(x_train, y_train,                #Trains model with 32 logs at a time
                    batch_size=32,
                    epochs=2,
                    verbose=1,
                    validation_split=0.1)
```

Code

```
score = model.evaluate(x_test, y_test,                #Evaluation of model
                       batch_size=32, verbose=1)
print('Test score:', score[0])
print('Test accuracy:', score[1])

collect_acc.append(score[1])
```

```
Found: 32 Categories

Train on 3600 samples, validate on 400 samples
Epoch 1/2
3600/3600 [==============================] - 1s 290us/step - loss: 0.4400 - acc: 0.8961 - val_loss: 0.1555 - val_acc:
0.9600
Epoch 2/2
3600/3600 [==============================] - 1s 173us/step - loss: 0.1057 - acc: 0.9722 - val_loss: 0.0774 - val_acc:
0.9825
1000/1000 [==============================] - 0s 62us/step
Test score: 0.05518945822119713
Test accuracy: 0.99
```