Uso de Autoencoders para detectar malware basado en la actividad de internet (Junio 2019)

Alejandro Ferraresso

***Abstract* — Existe gran variedad de técnicas y modelos que permiten detectar la presencia de malware en una red de computadores, analizando el tráfico que produce cada computadora. La mayoría de estas técnicas se basan en reglas o en modelos supervisados que necesitan tener actividad de diferentes malwares identificadas para poder ser utilizados. En éste trabajo se propone usar autoencoders[[1]](#footnote-1), una estructura particular de redes neuronales[[2]](#footnote-2), para poder detectar sin un dispositivo fue infectado por un malware[[3]](#footnote-3) basado en su tráfico de internet.**

***Palabras clave******s* - Autoencoders, red de computadoras, malware, control de trafico de red, redes neuronales.**

# Introduccion

Constantemente aparecen nuevos malwares, que se aprovechan de algún defecto de los programas o sistemas operativos para tomar control, parcial o total, de computadoras y luego buscan propagarse a otros dispositivos o los utilizan para realizar ataques masivos, como el de DDoS[[4]](#footnote-4). Todas esas actividades que suelen realizar lo malwares, implican un cambio en la forma que ese dispositivo utiliza su acceso a internet, cambiando los patrones de comportamiento, los requerimientos que hace, la distribución de los mísmos. En éste trabajo se propone utilizar un tipo de redes neuronales, llamados autoencoders, que pueden detectar las correlaciones en los datos de entrada al buscar una representación reducida de los mísmos. La idea de éste trabajo es que los patrones que el autoencoder aprenda cuando la computadora no está infectada, no serán los mismos que tendran los datos de la computadora, una vez infectada. Esto causará que el error de reconstrucción de los datos será mayor, permitiendo detectar que la computadora ha sido infectada. Primero se hará una breve explicación de autoencoders, mostrando los beneficios que tienen para ésta tarea. Luego se explicarán los datasets utilizado para la prueba, seguido de un breve detalle sobre el procesamiento de los mísmos. Al final del trabajo se explica el modelo propuesto y los resultados obtenidos.

# Autoencoders

Los redes neuronales autoencoders son una técnica de aprendizaje no supervisado[[5]](#footnote-5), que trata de aprender una forma de reconstruir el dato original, es decir, intenta aprender una aproximación de la función identidad[[6]](#footnote-6). Si bien la función identidad es claramente algo simple de aproximar, ésta tarea puede hacerse compleja si aplicamos algunas restricciones al modelo.

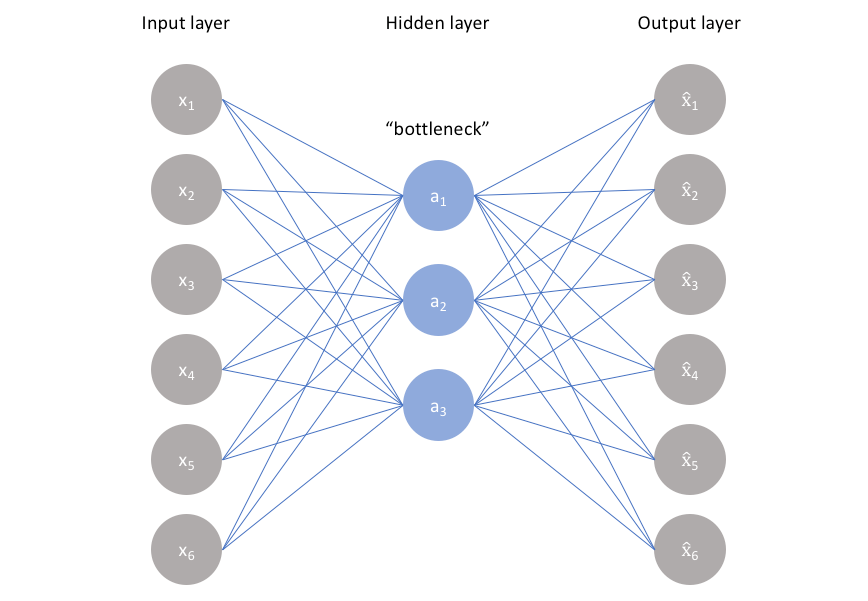


Figura 1

La principal restricción que se le impone a los autoencoders es la existencia de un *cuello de botella*, éste cuello de botella obliga a la red neuronal a aprender una representación comprimida de los datos originales, que permita volver a reconstruirlos. En la figura 1 se ilustra de forma simple éste cuello de botella, se tienen 6 neuronas de entrada, con una capa oculta de 3 neuronas, para luego obtener tantas neuronas de salida, como hubo de entrada, o sea que son 6. En autoencoders más complejos pueden existir más capas ocultas intermedias, tanto entre la entrada y el *cuello de botella*, como entre el cuello de botella y la salida. A los autoencoders suelen dividirlos en dos partes, codificador (encoder) y decodificador (decoder) como se muestra en la figura 2. El codificador se denomina de esa forma porque comprende todas las capas desde la entrada hasta el cuello de botella, es decir las capas que *codifican* la entrada a su representación reducida. Con el mismo criterio, el decodificador comprende todas las capas desde el *cuello de botella* hasta la capa de salida, es decir, las encargadas de *decodificar* la información comprimida y obtener el dato original.

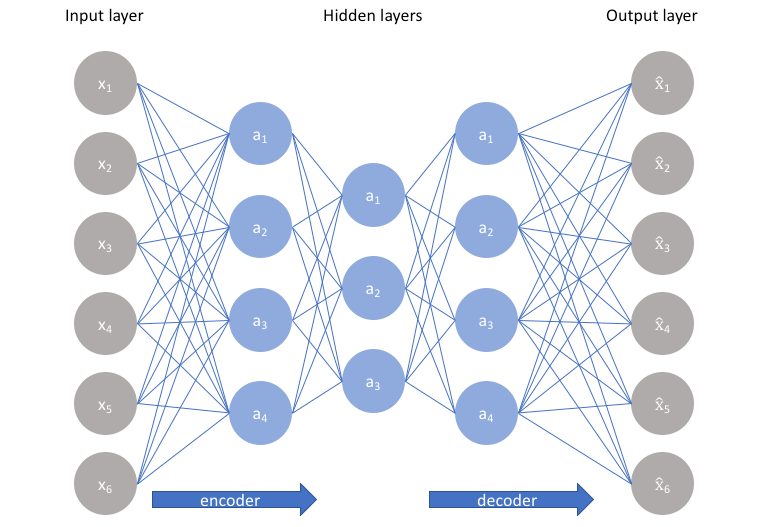


Figura 2

Si asumimos que las diferentes variables de entrada son completamente independientes entre sí, la tarea de conseguir una forma reducida de representar los datos sería altamente complicada. Sin embargo, si existe una cierta estructura en los datos, con relaciones entre las variables, esa estructura puede ser aprendida, y por lo tanto puede ser aprovechada al forzar a la red neuronal a pasar por el *cuello de botella*. Si nosotros quitáramos el *cuello de botella*, la red simplemente pasaría los valores de entrada a través de ella directamente hacia la salida. Al ponerle un *cuello de botella* limitamos la cantidad de información que pasa a través de la red, obligando a que encuentre formas de relacionar los valores de entrada, es decir que, a encontrar la estructura que los datos tienen.

Dado que las redes neuronales pueden aprender relaciones no-lineales, este mecanismo resulta en una generalización más potente de PCA[[7]](#footnote-7). Mientras que PCA intenta encontrar un hiperplano de menos dimensiones que pueda describir a los datos original, los autoencoders aprenden a representarlos en una superficie no lineal (Figura 3). De hecho, si en todas las capas de la red neuronal utilizamos únicamente funciones de activación lineales, obtendremos resultados similares a los de PCA[[8]](#footnote-8).

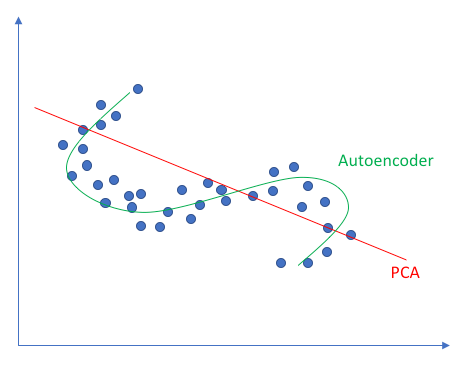


Figura 3

# Datasets

El dataset utilizado fue publicado por la gente de Stratosphere Lab[[9]](#footnote-9), quienes se encuentran desarrollando un IPS[[10]](#footnote-10). Ellos publican varias capturas del tráfico en una red controlada. Las capturas están divididas en tres grupos: normales, con malware y mixtas, todas hechas en ambientes controlados. Como sus nombres lo indican, las primeras corresponden a capturas hechas en situaciones normales, sin presencia de malware, las segundas corresponden a capturas hechas en ambientes que fueron infectados con algún malware y la última categoría corresponde a capturas hechas en ambientes que alternan su estado entre limpios e infectados con un malware. Para nuestro trabajo utilizamos una captura que pertenece a esta última categoría (la Mixed Capture 1), obtenido en una máquina virtual con Windows 7, que está dividida en 3 etapas:

1. La primer etapa consiste en una captura que dura aproximadamente un día. Durante éste tiempo, un usuario realiza acciones comunes en la computadora, descarga e instala skype, chatea con algunas personas e intenta hacer una llamada. Accede a cuentas de mail y a redes sociales como Facebook, juega al Candy Crush de Facebook. Luego mira videos en Youtube, y utiliza Google Drive para subir algunos documentos. Finalmente lee noticias en diarios online, realiza algunas búsquedas y finaliza la captura. Luego de guardar la captura, copia el malware al windows dejando infectada la máquina virtual.
2. Una vez que está infectada, el usuario continúa haciendo acciones normales, es decir, vuelve a leer diarios, utiliza Facebook, Skype, Gmail, Dropbox, realiza búsquedas en Google. Pero mientras va realizando éstas acciones, el malware le va ofreciendo diferentes avisos y publicidades a los que en algunos casos el usuario accede y en otros no. Esta captura dura también un día.
3. En la tercer captura, el usuario primero hace unas búsquedas en google para descargar software de diferentes páginas que le permitan borrar el malware, se realizan varios reinicios en el proceso. Una vez que la pc esta limpia, es decir, que ya no posee malware corriendo, el usuario retoma sus actividades normales, leyendo diarios, utilizando Gmail y Facebook, mirando videos en Youtube. Esta última captura es más corta que las demás, se extiende por 17 horas, aproximadamente.

Se eligió esta captura por dos motivos principales. En primer lugar, presenta un orden cronológico con etapas bien marcadas y diferenciadas, permitiéndonos simular en la evaluación el comportamiento normal de un usuario. Segundo, la distribución de las tres etapas nos permite validar si el modelo nos permite diferenciar el momento en que la máquina es desinfectada, además de detectar el momento en que la máquina deja de comportarse normalmente debido a que fue infectada.

# Pre-procesamiento de los datos

Los datos de la captura están divididos en 3 archivos:

1. 2015-07-28\_mixed.binetflow.before.infection para antes de la infección.
2. 2015-07-28\_mixed.binetflow.after.infection con la captura durante la infección.
3. 2015-07-28\_mixed.binetflow.after.desinfection con el proceso de desinfección y el uso hasta apagar la máquina.

Los tres archivos son de texto plano con la información de los flujos Argus[[11]](#footnote-11), incluyendo los primeros 480 bytes de los paquetes enviados en el flujo. Los archivos son similares a un CSV, aunque pueden incluir alguna coma producto de los 480 bytes de los paquetes, por lo que el primer paso fue borrar esas comas extra para poder procesar los archivos como CSV. Por cada flujo, los archivos poseen varias variables, de las que se usaron las siguientes:

* StartTime: Dia y hora del flujo.
* Dur: Duración.
* Proto: Procolo (tcp, udp, etc)
* SrcAddr: IP origen.
* Sport: Puerto origen.
* Dir: Dirección del flujo (subida, bajada, ambos)
* DstAddr: IP destino.
* Dport: Puerto destino.
* TotPkts: Total de paquetes
* TotBytes: Total de bytes.

Dado que la idea era simular una situación real, era necesario darle forma a los datos de forma que sirviera para darle un contexto de comportamiento a lo largo de cierto tiempo y de forma que el modelo lo pudiera entender. Para ésto se decidió que en lugar de darle directamente la información de los flujos al modelo, armar por cada uno de ellos una serie de datos que describieran los sucedido en cierta ventana de tiempo anterior al flujo. Entonces, por cada flujo que había en las capturas, se buscaban todos los flujos que había dentro de la ventana de tiempo, y se realizaron algunos cálculos sobre las variables mencionadas (como calcular la media, la moda, la desviación estándar, contar la cantidad de flujos, etc), diferenciando si el flujo era de subida o en ambos sentidos, descartando los demás sentidos porque aparecían muy poco en los datasets. De ésta forma se obtuvieron 3 nuevos datasets, cada uno con 55 columnas.

# Modelo propuesto

Se propone utilizar una red neuronal de tres capas completamente conectadas, como muestra la figura 4. El modelo fue entrenado para reducir el error cuadrático medio[[12]](#footnote-12) (mean squared error, en ingles) entre el dato reconstruido y los datos reales.

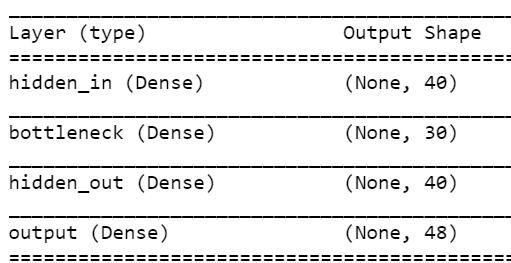


Figura 4

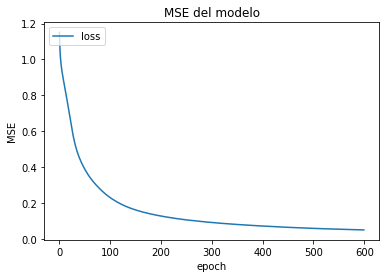


Figura 5

Para entrenar se dividió el dataset anterior a la infección, dejando fuera del entrenamiento la última hora (3264 registros), para poder usarlo para validar el modelo, de forma de tener datos de validación antes y después de la infección.

Se realizó el entrenamiento usando Stochastic Gradient Descent (SGD)[[13]](#footnote-13) durante 600 iteraciones sobre todos los datos de entrenamiento. La figura 5 muestra como se fue reduciendo el error en cada iteración, hasta que se llegó al valor de 0.05.

# Resultados

La figura 6 muestra el error de la reconstrucción del dato a lo largo de los ejemplos de los tres datasets, ordenados cronológicamente. La línea vertical verde indica el momento en que se termina el conjunto de entrenamiento y comienzan los datos de validación (ambos pertenecientes al dataset anterior a la infección). Luego de éso, el espacio contenido entre ambas líneas rojas representa el intervalo de tiempo que la máquina estuvo infectada por el malware. Lo que se encuentra luego de la segunda línea roja es lo que corresponde al tercer dataset, cuando se remueve el malware y se vuelve a utilizar normalmente.

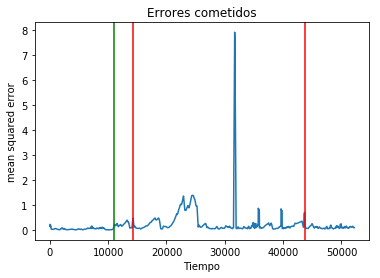


Figura 6

Para facilitar la visualización del gráfico se utilizó la media móvil simple con un ventana de 200 datos.

Claramente se puede notar un aumento del error en varios intervalos de tiempo, incluyendo algunos de mucho error. Aunque ese aumento de error no se percibe luego de que se elimina la infección, lo que indica que el error no se debe a la distancia en tiempo desde en conjunto de entrenamiento.

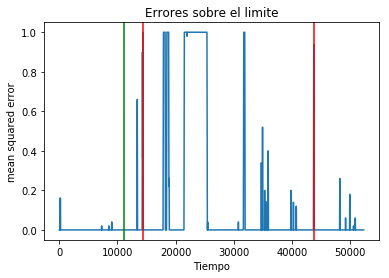


Figura 7

También se optó por hacer un análisis contemplando la cantidad de veces que se supera un cierto límite de error, sin tener en cuenta el margen del error, para tratar de determinar si además de presentar errores con valores más altos, durante la infección se presenta más cantidad de errores. Para ésto, se planteó un límite basado en la media de error sobre el conjunto de entrenamiento, asignando valor uno si se supera dicho límite y cero en caso contrario. En la figura 7 se muestra el comportamiento en los diferentes intervalos, nuevamente usando una media móvil para facilitar la visualización. Si bien usando ésta forma se aprecian errores, tanto en el período de validación como luego de la desinfección, hay una clara diferencia con el intervalo de la infección que presenta muchos más valores superando el límite. La siguiente tabla muestra la media de error y la media superando el límite de los 4 conjuntos de datos. Se nota nuevamente una clara diferencia en el caso del intervalo infectado.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Entrena-miento | Valida-  ción | Infectado | Desinfec-tado |
| Media de error | 0.05 | 0.22 | 0.33 | 0.12 |
| Media por límite | 0.001 | 0.02 | 0.17 | 0.01 |

Tabla 1

# Conclusion

En éste trabajo se planteó la posibilidad de utilizar un modelo no supervisado de redes neuronales para detectar si una máquina fue infectada por malware, basado en comportamiento previo. Se mostró que es posible entrenar un modelo así, que aunque no produjo resultados perfectos, son muy prometedores.

Aunque se usó un período de entrenamiento muy acotado, menor a un día, el modelo presentado marcó claramente un aumento en el error durante la infección, y a continuación se redujo el error al ser desinfectada la máquina. Si bien la diferencia entre ambos estados no es constantemente alta, esto puede deberse a la falta de datos de entrenamiento, o a que el malware no se comporta todo el tiempo de la mísma forma. Incluso queda para analisis future mejorar el preprocesamiento de los datos, que se hizo de una forma bastante simple.

Todo ésto nos indica que es posible utilizar soluciones de éste estilo para controlar redes de computadoras de forma no supervisada, lo que disminuye el trabajo necesario para la puesta en marcha y permite adaptarlos automáticamente a lo largo del tiempo.

1. "Autoencoders - Unsupervised Feature Learning and Deep Learning ...." <http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-1)
2. "Redes Neuronales, ¿qué son?. – William Khepri – Medium." 2 nov.. 2018, <https://medium.com/@williamkhepri/redes-neuronales-que-son-a64d022298e0>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-2)
3. "Malware - Wikipedia." <https://en.wikipedia.org/wiki/Malware>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-3)
4. "Ataque de denegación de servicio - Wikipedia, la enciclopedia libre." <https://es.wikipedia.org/wiki/Ataque_de_denegaci%C3%B3n_de_servicio>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-4)
5. "¿Aprendizaje supervisado o no supervisado? – Juan Zambrano ...." 30 mar.. 2018, <https://medium.com/@juanzambrano/aprendizaje-supervisado-o-no-supervisado-39ccf1fd6e7b>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-5)
6. "Función identidad - Universo Formulas." <https://www.universoformulas.com/matematicas/analisis/funcion-identidad/>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-6)
7. "Principal Component Analysis." <ftp://statgen.ncsu.edu/pub/thorne/molevoclass/AtchleyOct19.pdf>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-7)
8. "How is a linear autoencoder equal to PCA? - Quora." 8 abr.. 2017, <https://www.quora.com/How-is-a-linear-autoencoder-equal-to-PCA>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-8)
9. "Stratosphere IPS." <https://www.stratosphereips.org/>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-9)
10. "What is an Intrusion Prevention System? - Palo Alto Networks." <https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-prevention-system-ips>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-10)
11. "ARGUS- Auditing Network Activity - QoSient." 14 ene.. 2019, <https://qosient.com/argus/>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-11)
12. "Error cuadrático medio - Wikipedia, la enciclopedia libre." <https://es.wikipedia.org/wiki/Error_cuadr%C3%A1tico_medio>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-12)
13. "ML | Stochastic Gradient Descent (SGD) - GeeksforGeeks." <https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/>. Se consultó el 9 jun.. 2019. [↑](#footnote-ref-13)