Confundiendo una red neuronal utilizando ejemplos adversarios

Edmundo Manuel Miranda Bailón
National University of Engineering
Lima, Perú
emirandab@uni.pe

Elias Josue Llampi Aliaga
National University of Engineering
Lima, Perú
elias.llampi.a@uni.pe

José Manuel Yoichi Rios Yamamoto
National University of Engineering
Lima, Perú
jriosy@uni.pe

***Resumen*—Muchas herramientas con tecnologías de aprendizaje automático integran redes neuronales, por lo cual se requiere probar su buen funcionamiento, precisión y seguridad ante posibles ataques. Estudios recientes descubrieron vulnerabilidades en la clasificación de imágenes de redes neuronales. Estas vulnerabilidades son denominadas ejemplos adversarios (*adversarial examples*): datos que al ser modificados la red neuronal no los clasifica correctamente.**

En este trabajo se realizará un estudio sobre los ejemplos adversarios al confundir una red neuronal entrenada para clasificar imágenes de dígitos del conjunto de datos MNIST. Se analizará la precisión de la red neuronal antes y después de los diferentes ataques mediante el método Fast Gradient Sign, teniendo como referencia un hiperparámetro ϵ el cual utilizaremos para crear las imágenes adversarias que dictará qué tan modificada será la imagen de entrada. Con esto se vera la disminución progresiva de la precisión del modelo.

Palabras claves— Ejemplos adversarios, ataques adversarios, redes neuronales, MNIST.

I. Introducción

En los últimos años, en el área del aprendizaje automático se han hecho numerosos avances. Estos avances se deben en gran parte a la disponibilidad de conjuntos de datos que se pueden utilizar para el entrenamiento de modelos, y al progreso de los procesadores de cómputo que ayudan a disminuir el tiempo de entrenamiento. El desarrollo continuo de Machine Learning y herramientas que emplean estas técnicas de aprendizaje automático ayudan a resolver problemas como: sistemas de videovigilancia, reconocimiento de voz, tareas de clasificación y análisis de sentimientos en autómatas, como otros. Por lo cual, es de interés mejorar este tipo de herramientas al utilizar Deep Learning, que es un tipo de Machine Learning en donde sus modelos contienen redes neuronales profundas, las cuales funcionan como una extensión de una red neuronal al añadir capas ocultas en su estructura, y así mejorar la precisión en cuanto a la predicción para obtener excelentes resultados en múltiples áreas.

Sin embargo, se ha demostrado recientemente que estas redes son vulnerables a los denominados ejemplos adversarios (*adversarial examples*).

Un ejemplo adversario es una entrada de datos que al ser modificado logra engañar o confundir a una red neuronal y deriva en una clasificación incorrecta.

Figura 1. Ejemplo adversario obtenido al añadir ruido a la imagen original para lograr confundir a la red neuronal y retorne como resultado un gibbon (Mono Gibón)

Aquí se puede observar que si se toma una imagen clasificada correctamente por la red neuronal, en este caso un panda, y se añade un poco de *ruido*, se consigue que la red lo confunda y lo clasifique de forma errónea al resultar un mono gibón en lugar de un panda [1].

En los objetivos de los ejemplos adversarios están las redes neuronales aplicadas a [4]:

- Visión por computadora: clasificación de imágenes, detección de objetos.
- Procesamiento de lenguaje natural: traducción automática, generación de texto.
- Seguridad del ciberespacio: servicios en la nube, detección de malware, intrusiones en la red.
- Físico: reconocimiento de señales de trafico, cámaras de suplantación, reconocimiento de rostros.

En este trabajo se analizará una red neuronal convolucional programada en el lenguaje Python y preentrenada bajo una base de datos MNIST, de forma que clasifique números de 28x28 píxeles [1] y luego se generarán ejemplos adversarios no dirigidos para evidenciar la vulnerabilidad de la red neuronal.

I-A. Objetivo General

Generar ejemplos adversarios para exponer la vulnerabilidad de una red convolucional (CNN) y utilizar el dataset de MNIST para estudiar su comportamiento.

I-B. Objetivos específicos

- Implementar el método de símbolo de gradiente rápido FGSM(**fast gradient sign method**) para agregar *ruido* a una imagen, que al introducirla a la red neuronal ocasionará una mala clasificación de los datos y se expondrá su vulnerabilidad.
- Evaluar la red neuronal ante distintos ataques al utilizar el método de símbolo de gradiente rápido FGSM(**fast gradient sign method**) con varios valores para el hiperparámetro "ϵ".

II. Fundamento teórico

II-A. Redes Neuronales

Una red neuronal funciona como la representación matemática de las neuronas en el cerebro, capaces de emular su funcionamiento en la ejecución de tareas complejas donde cada entrada pasa por diferentes capas, puede ser directo a la capa de salida o pasar por capas intermedias, mejor conocidas como *capas ocultas*, donde a cada capa se aplica una función matemática para determinar su salida a la siguiente capa o clasificación.

Las redes neuronales cuentan con una función de activación y un conjunto de pesos, si el conjunto de pesos va de un valor positivo a uno negativo, entonces se requiere de una transformación de todos los pesos para que la función de activación se ajuste a esto. Una función de activación es la *función sigmoide*, encargada de hacer esta transformación.

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

La función sigmoide es aplicada a la suma del producto de pesos y sus activaciones, donde $w_1, w_2, ..., w_n$ son los pesos y $a_1, a_2, ..., a_n$ sus activaciones.

$$\sigma(w_1 a_1 + w_2 a_2 + ... + w_n a_n) \tag{2}$$

Si se desea que una neurona en particular se active o no para un valor de su función de activación se requiere de un sesgo el cual determinará si las neuronas están encendidas o no. Con lo cual para obtener una red neuronal convencional propiamente dicha, el sistema debe hallar los pesos y sesgos correctos. Sea 0 la capa de entrada y 1 la primera capa oculta, entonces

$$a^{(1)} = \sigma(wa^{(0)} + b) \tag{3}$$

Donde w son los pesos $a^{(0)}$ y $a^{(1)}$ son las activaciones de la capa 0 y capa 1, respectivamente, y b el sesgo.

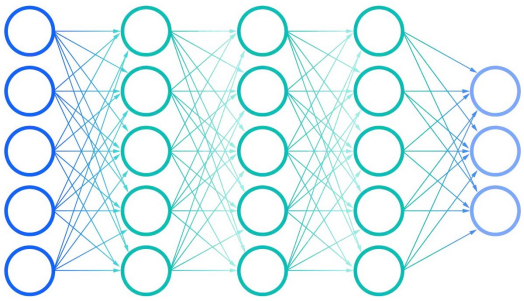

Figura 2. Ejemplo de una red neuronal con 3 capas ocultas

II-B. Aprendizaje de la red

La función de costo calcula la suma de cuadrados de la diferencia entre la salida de activación de la predicción y la salida de activación correcta. En la capa final se busca que la neurona a activarse tenga una salida de 1.

Para converger a un mínimo local en la función de costo y que no se disparen los pesos y sesgos a valores muy altos, se calcula la gradiente descendiente. Los ajustes a algunos pesos tendrán mayor efecto en la función de costo que ajustar otros pesos. Se halla la derivada de la función de pérdida, la cual es la pendiente de la función en un punto dado. Por ejemplo, en un caso de reconocimiento facial si se falla en reconocer la imagen, entonces se tienen que ajustar los pesos y sesgos para poder lograr que la neurona de salida que debe activarse, se active con la imagen dada y no se cometa el mismo error. Asimismo con el incremento de la función de activación de la neurona deseada, la activación de las demás neuronas debe decrecer. Este proceso es llamado *Propagación hacia atrás (Backpropagation)* y es realizado por todo elemento dentro de los datos de información. Debido a que este proceso consume tiempo, se divide los datos de entrenamiento en grupos y se calcula en cada uno de los grupos cada paso, por esto se usa la *Gradiente Estocástica* para el mínimo local [6].

II-C. Redes Neuronales Convolucionales (CNN)

Las redes neuronales son redes totalmente conectadas, debido a esto se vuelve inadecuado para el proceso de imágenes como clasificación y reconocimiento. La explicación a esto es debido a que al tomar una imagen, su tamaño de análisis es de $mxnxc$ donde m y n se refiere a los números de píxeles por fila y columna y c se refiere a los números de canales que puede tomar este píxel como color; por ejemplo, una red entrenada en imágenes de MNIST toma imágenes de 28 x 28 y debido a que es escala de grises solo tiene un canal por lo que se tiene una entrada de $28x28x1 = 784$ pesos por neurona. Y si queremos algo como reconocer un rostro se tendría una imagen de $230x230$ y como se tiene los canales RGB tendríamos $230x230x3 = 158700$ pesos por

neurona en la capa de entrada, esto teniendo en cuenta que las capas ocultas también cuentan con mas de una neurona, el cálculo se vuelve costoso al tener en cuenta que la red esta totalmente conectada.

Las redes neuronales convolucionales *(CNN)* son redes neuronales con múltiples capas ocultas. Su diferenciación se basa en crear filtros para las imágenes en lugar de analizar píxel por píxel como hace tradicionalmente una red neuronal, esto se logra con una estructura la cual no esta totalmente conectada, una neurona puede estar conectada a solo un par de neuronas del total en la capa posterior [4].

La estructura de una CNN consiste en la siguiente formación:

- Convolución
- Rectificación de unidad lineal
- Pooling
- Totalmente conectado

Las imágenes de un mismo número escrito a mano puede presentar diferencias entre una imagen y otra. En estos casos es necesario que la CNN pueda identificar el mismo número en todas las imágenes disponibles.

Filtros o características o neuronas son seleccionadas de la imagen base, de esta forma se hace más factible el reconocimiento de la imagen. Los filtros son partes de la imagen que son elegidas para identificar otras imágenes que quizá pertenezcan a la misma clasificación, por ejemplo, en un número, una curva puede diferenciar un 8 de un 1. Una imagen puede tener varios filtros y estos son usados para construir la capa convolucional.

Los pasos para la construcción de esta capa son los siguientes

1. Multiplicar cada píxel en la imagen con el píxel correspondiente en el filtro escogido
2. Agregar los productos obtenidos
3. Dividir la suma por el número total de píxeles en el filtro

Mover el filtro a través de la imagen y repetir los pasos para todos los píxeles en la imagen. La salida es la convolución de la imagen con cada otro filtro seleccionado al inicio del proceso, esto hace que la selección de filtros tome un papel crucial en el rendimiento de una CNN. Cabe recalcar que el término *convolución* en CNN indica encontrar una posible coincidencia al aplicar el filtro recursivamente por la imagen, o mejor dicho, a través de cada píxel. Con esto tenemos que una *capa convolucional* es la pila de estas imágenes filtradas.

El proceso de *Pooling* es hecho para todas las imágenes filtradas en la capa convolucional para obtener una imagen achicada. El pooling reduce el tamaño de la imagen al solo seleccionar los píxeles mejor evaluados en una ventana de tamaño predefinido . El resultado final del pooling es una pila reducida de imágenes. Este paso ayuda a a reducir el mapeo de la imágenes.

Otro paso es la normalización, esto es hecho al cambiar los valores negativos del píxel en una imagen a cero, lo que nos devuelve una nueva pila de imágenes sin valores negativos, esto es conocido como la *capa de Rectificación de Unidad Lineal*. Así, esta capa activa una neurona solo si un píxel de la imagen de entrada esta arriba de un umbral particular. Esto ayuda a remover los ceros de las imágenes en la capa convolucional. Posteriormente hacemos pooling, Convolución y Rectificación de Unidad Lineal y obtenemos una sola pila y la salida de una se vuelve la entrada de la otra.

Estos 3 pasos se repiten cuantas veces sean necesarios. El paso de imágenes a través de diferentes capas convolucionales resulta en imágenes mucho mas filtradas, mientras que imágenes pasadas a través de pooling resulta en imágenes cada vez más pequeñas. La capa final es una capa totalmente conectada donde cada neurona y cada valor de píxel puede apoyar en determinar el valor de salida. Se pueden determinar valores altos en píxeles para un cierto tipo de imagen de entrada así como valores bajos en píxeles para otro tipo de imágenes de entrada, estos píxeles que se determinan como altos para una imagen indican que tan fuerte será la predicción de la clasificación de la imagen, esta predicción se obtiene al pasar la imagen por la CNN lo que consiste en pasarla por todas las capas y basado en los valores de los píxeles de la salida de la capa final y sus correspondientes pesos, se determinará la clase a la cual pertenece la imagen de entrada [6].

A continuación se presenta la estructura de la CNN a la cual se apunta en este trabajo.

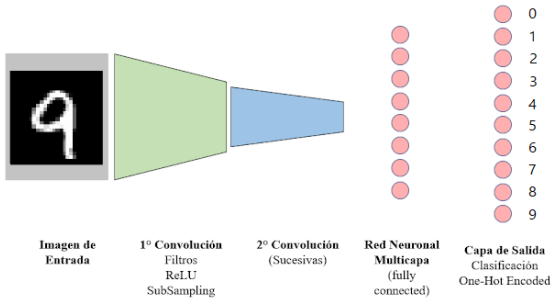

Figura 3. Red Neuronal Convolucional

II-D. Ejemplos adversarios

Los ejemplos adversarios son muestras de datos de entrada a los que se añaden pequeñas perturbaciones irreconocibles al ojo humano y que provocan una clasificación errónea en el modelo objetivo. [4].

Si se considera un modelo M y una muestra x, que es clasificada correctamente por el modelo: $M(x) = y_{true}$. Al agregar una ligera perturbación a x, se puede crear una muestra adversaria x', que es similar a x pero que es clasificado erróneamente por el modelo: $M(x') \neq y_{true}$ En la Figura I, el modelo considera la imagen original como un "panda" con 57,7 % de precisión. Al agregar a la imagen una pequeña perturbación, se clasifica como

un "gibón"por el mismo modelo con 99,3 % de presición, mientras que los ojos humanos no pueden identificar la diferencia.

II-D1. Capacidades adversarias:

El término **capacidades adversarias** se refiere a la información que se puede obtener sobre los modelos objetivos y usar dicha información para realizar ataques. Mientras mayor acceso a la información se posea para un adversario, más $fuerte$ será.

Las *capacidades adversarias* se clasifican en:

1. **Nivel de entrenamiento:** Los ataques en este nivel modifican o corrompen el conjunto de datos que es usado por un modelo para su entrenamiento. Se encuentra lo siguiente:
   - **Inyección de datos:** se corrompe el modelo objetivo al ingresar muestras adversarias al conjunto de datos de entrenamiento.
   - **Modificación de datos:** se corrompe el modelo objetivo al modificar el conjunto de datos de entrenamiento antes de ser usado por el modelo.
   - **Corrupción lógica:** se tiene acceso a los algoritmos de aprendizaje del modelo.
2. **Nivel de prueba:** En este nivel se obliga al modelo, que ya ha sido entrenado a producir resultados incorrectos. La efectividad de los ataques dependen de la cantidad de información disponible sobre el modelo objetivo. Se divide en:
   - **White-box:** En los ataques de caja blanca, se posee total conocimiento sobre el modelo objetivo: algoritmos de entrenamiento, distribución de datos y parámetros.
     Se identifican vulnerabilidades con dicha información y se alteran los datos de entrada con algún generador de ejemplos adversarios.
     En este trabajo se empleará el generador de ejemplos adversarios **FSGM (Fast Gradient Sign Method)**
   - **Black-box:** En estos ataques no se posee información sobre el modelo objetivo. En cambio, se analizan las vulnerabilidades del modelo al usar un registro de pares de *entrada/salida* del modelo. Es decir, se ingresa al modelo una serie de muestras adversarias y se observan las correspondientes salidas.

II-E. Fast Gradient Sign Method (FGSM)

FGSM es un método para generar ejemplos adversarios, fue introducido por Goodfellow en el paper **Explaining and Harnessing Adversarial Examples** [1]. Este método trabaja del siguiente modo:

1. Tomar una imagen de entrada.
2. Realizar una predicción con dicha imagen y el modelo objetivo entrenado.
3. Calcular la función de pérdida basado en la clasificación correcta o incorrecta de la imagen.
4. Calcular el signo del gradiente.
5. Usar el signo del gradiente para construir el ejemplo adversario.

Se construye entonces la fórmula del siguiente modo:

$$x' = x + \epsilon.sign(\nabla_x J(\theta, x, y)) \tag{4}$$

Donde x es la imagen de entrada, x' es la imagen modificada, ∇ es el gradiente de la función de costo J respecto a X, con parámetros θ y etiqueta de salida y. Además, el término ϵ es un número pequeño, que puede ser ajustado a conveniencia, cabe resaltar que mientras más aumente en valor ϵ, la imagen resultante x' se hará cada vez más diferente de la imagen original ante los ojos humanos.

El término η, correspondiente a:

$$\eta = sign(\nabla_x J(\theta, x, y)) \tag{5}$$

es el ruido que se añade a la imagen original x, para generar un ejemplo adversario x'.

III. Estado del Arte

III-A. Intriguing properties of neural networks

Los ejemplos adversarios, presentados por primera vez en el año 2014 en el artículo **Intriguing properties of neural networks** [5] , mostraron la inestabilidad de las redes neuronales con respecto a pequeñas perturbaciones en los datos de entrada.

Al considerar una red neuronal de ultima generación que generalice bien una tarea de reconocimiento de objetos, se esperaba que una pequeña perturbación no podría cambiar la categoría de una imagen. Sin embargo, se encontró que al aplicar una perturbación imperceptible no aleatoria a una imagen de prueba, era posible arbitrariamente cambiar la predicción de la red. Dichas perturbaciones se encontraron maximizando el error de predicción. Además, también se encontró que los ejemplos adversarios eran robustos, es decir, si se usa una red neuronal para entrenar un conjunto de ejemplos contradictorios, se encuentra que estos ejemplos siguen afectando a otras redes neuronales, incluso cuando se entrenaron con diferentes parámetros o más aún, con un conjunto diferente de datos.

III-B. Explaining and Harnessing Adversarial Examples

Goodfellow, Shlens y Szegedy [1] parten de que las estrategias de regularización genéricas, como la deserción, la capacitación previa y el promedio de modelos,

no confieren una reducción significativa en la vulnerabilidad de un modelo a los ejemplos adversarios, pero en familias de modelos no lineales como las redes RBF, pueden hacerlo. En el artículo se sugiere una tensión entre el diseño de los modelos que son fáciles de entrenar debido a su linealidad y al diseño de modelos no lineales para resistir las perturbaciones adversas.
De este modo, para los modelos lineales, la facilidad de entrenamiento del modelo implicaría una mayor vulnerabilidad a los ejemplos adversarios, mientras que para los modelos no lineales, cuyo entrenamiento es más costoso, la resistencia a los ejemplos adversarios sería mayor.

III-C. Adversarial examples are not bugs, they are features

En el artículo **Adversarial Examples Are Not Bugs, They Are Features** [2] se demuestra que los ejemplos adversarios se pueden atribuir directamente a la presencia de características no robustas: características (derivadas de patrones en la distribución de datos) que son altamente predictivas, pero frágiles y (por lo tanto) incomprensibles para los humanos. Ya que generalmente se entrenan a los clasificadores para maximizar únicamente la precisión. En consecuencia, los clasificadores tienden a usar cualquier señal disponible para hacerlo, incluso aquellas que parecen incomprensibles para los humanos.

Además, se menciona que las perturbaciones adversarias calculadas para un modelo a menudo se transfieren a otros modelos entrenados de forma independiente. Ya que es probable que dos modelos aprendan características no robustas similares, por lo que las perturbaciones que manipulan dichas características se aplicarán a ambos.

Finalmente, se establece la vulnerabilidad adversaria como un fenómeno centrado en el ser humano. Ya que, desde el punto de vista del aprendizaje supervisado estándar, las características no robustas pueden ser tan importantes como las robustas.

III-D. Review of Artificial Intelligence Adversarial Attack and Defense Technologies

El artículo **Review of Artificial Intelligence Adversarial and Defense Technologies** [4] realiza una recopilación y clasificación de los métodos de ataques adversarios hasta el año 2019, así como los métodos de defensa que mejores resultados han mostrado.

Para este artículo la atención se centró en un tipo de ataque en particular.

En el artículo se menciona un tipo de ataque llamado ataque de caja blanca, donde se dispone de los datos del modelo objetivo, como parámetros, algoritmos, y estructura del modelo. Usando esta información se crean los ejemplos adversarios mediante funciones generadoras que reconstruyen la función de error. También se muestra en el artículo que el uso de redes adversarias generativas (GAN), obstaculizan la reconstrucción de la función del error. Sin embargo, sin el entrenamiento adecuado de estas redes GAN, el rendimiento disminuye, por lo que resulta desafiante.

IV. Revisión de la literatura

IV-A. Cadenas de búsqueda

Se realizó la búsqueda de trabajos y documentos relacionado a los ejemplos adversarios en redes neuronales en el buscador Google Académico y páginas web como Scopus, Arxiv, ResearchGate y Github. Realizando las siguientes consultas.

- TITLE-ABS-KEY ((Neural Network) AND (Fast gradient sign method))
- TITLE-ABS-KEY ((Neural Network) AND (Adversarial examples))
- TITLE-ABS-KEY ((Neural Network) AND (Adversarial images))

IV-B. Preguntas de Revisión

- ¿Cómo se generan los ejemplos adversarios en una red neuronal?
- ¿Cómo generar imágenes adversarias utilizando el método de signo de gradiente rápido?

Los resultados utilizando cadenas de búsquedas y las preguntas de revisión nos proporcionan la base para explicar y realizar la metodología que se presenta en la siguiente sección.

V. Metodología

En esta sección se describen las herramientas y la metodología que se usará en el presente trabajo:

V-A. Pytorch.torchvision

Torchvision es una biblioteca para Computer Vision que va de la mano con PyTorch que es un machine learning framework de código abierto. Tiene utilidades para transformaciones eficientes de imagen y video, algunos modelos pre-entrenados de uso común y algunos conjuntos de datos. El paquete de torchvision consta de datasets populares, modelos y transformaciones de imágenes comunes (torchvision.transforms) para Computer Vision. En este trabajo se utilizará el conjunto de datos MNIST (Modified National Institute of Standards and Technology database).

V-B. Pytorch.nn

Pytorch.nn es un paquete que contiene multitud de clases que nos permiten crear de una manera intuitiva redes neuronales y a la vez tener un nivel de detalle y control de los componentes de las mismas

V-C. Pytorch.optim

Pytorch.optim es un paquete que implementa varios algoritmos de optimización que se pueden implementar fácilmente. Los métodos comúnmente utilizados ya son compatibles, por lo que no hay ninguna necesidad de crearlos desde cero.

V-D. Generación de Adversarios

Debido a que el comportamiento y construcción de la red neuronal es lineal, se puede aprovechar esta característica para poder confundirla, esto es debido a que la linealidad permite una optimización más rápida, entonces una perturbación lineal en la entrada puede ocasionar problemas para la red neuronal.

A continuación se presenta el mismo ejemplo de la Figura 1 pero ahora con sus respectivas formulas de perturbación.

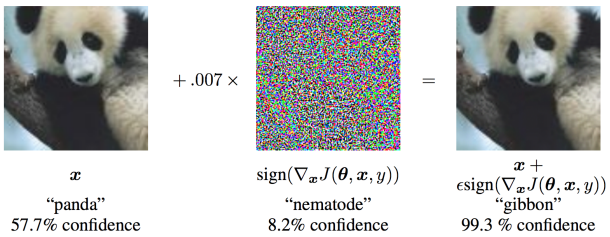

Figura 4. Formula de perturbación agregada

Se puede observar que mediante la suma de un vector cuyos elementos son los signos de los valores de la gradiente de la función de costo para esa respectiva entrada multiplicado por un hiperparámetro "ϵ", en el ejemplo se usa $\epsilon = 0{,}007$, se consigue confundir a la red neuronal y obtener una clasificación errónea.

Se tomará la siguiente ecuación:

$$\eta = \epsilon sign(\nabla_x J(\theta, x, y)) \tag{6}$$

Siendo "θ" los parámetros del modelo, "x" la imagen de entrada al modelo, "y" los objetivos asociados a la entrada "x" (opcional si lo hubiera) y la función "$J(\theta, x, y)$" el costo de entrenar la red neuronal, con esto se obtiene la función para conseguir la perturbación y solo queda probar que tan fuerte se quiere que sea esta perturbación mediante "ϵ".

Al usar su mismo método de entrenamiento, ya que se usan las gradientes halladas en el paso de Backpropagation para luego crear los ejemplos adversarios, se conseguirá una forma sencilla de confundir a la red neuronal. esto se refiere a la generación de ejemplos adversarios mediante **fast gradient sign method**. [1]

V-E. Métricas

En el presente trabajo se usará el hiperparámetro ϵ como métrica para la tarea de evaluar el efecto del ataque por FGSM en la red neuronal.

V-E1. Epsilon(ϵ): Tal como se presentó en la Figura 4, mediante el uso de las gradientes de aprendizaje se obtiene el vector el cual podemos sumar al vector de la imagen, donde esta valorizado cada pixel de los 28 x 28 pixeles.

Si sumáramos estas gradientes al vector de la imagen conseguiríamos una imagen llena de ruido y se volvería irreconocible y ese no es el objetivo de un ejemplo adversario ya que aunque su propósito es engañar a la red neuronal, también debe ser capaz de engañar al ojo humano en una aplicación real de estos ataques. Por esta razón este vector es multiplicado por un hiperparámetro ϵ cuyo rango es de $[0, 1]$, con esto obtenemos una forma de controlar el ruido que se agrega a la imagen.

$$ruido = \epsilon sign(\nabla_x J(\theta, x, y))$$

El valor de ϵ se determina por el programador a medida de evaluación para saber cuanto afecta el ataque a la precisión del modelo. En este trabajo se utilizarán los valores de 0.05, 0.1, 0.15, 0.2, 0.25, 0.3 para ϵ.

V-F. Metodología de trabajo

El proceso a seguir para la implementación será la siguiente:

1. Construcción de la red neuronal convolucional, que toma como estructura el modelo convolucional de Pytorch/examples/mnist
2. Importación del modelo pre-entrenado 'MNIST-Network.pth' [3] y carga de los datos de prueba que se usarán para la generación de ejemplos adversarios.
3. Mediante una función de rendimiento en los datos de prueba, se obtendrán las gradientes de las imágenes procesadas, se aplicará la formula (4) y se generaran las imágenes con ruido agregado mediante FGSM.
4. Los ejemplos adversarios generados se ingresarán al modelo objetivo como entradas para evaluar la presición del modelo con estos datos.
5. Se verificará que la presición del modelo con el conjunto de muestras adversarias es menor que la presición del modelo con los datos de entrada originales.

VI. Experimentación y Resultados

Como entorno de desarrollo se implementó el código en Google Colab con uso de la CPU.

VI-A. Datos

Los datos de prueba y sus etiquetas para evaluar la red neuronal y la generación de ejemplos adversarios fueron importados mediante el uso de las librerías *Datasets* y *DataLoader* de Pytorch

De la misma manera se hizo uso de la librería *Transforms* de Pytorch para almacenar los datos en tensores.

VI-B. *Ejemplos adversarios*

Al evaluar la red neuronal con las imágenes importadas sin modificar se obtuvo una precisión de 98.10 % del total de 10 000 con una pérdida promedio de 0.2393.

El objetivo fue generar ejemplos adversarios al tomar distintos valores para ϵ, de esta forma se pudo ver su influencia en la cantidad de ruido que se agregó después de tomar como base las gradientes de la imagen entrante.

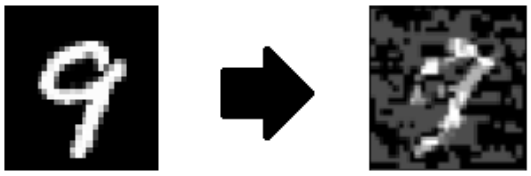

Figura 5. Imagen generada con un $\epsilon = 0.3$

Para el experimento se usaron los valores de 0.05, 0.1, 0.15, 0.2, 0.25, 0.3 para ϵ así como también el valor de 0 para tener una comparación de la clasificación de las imágenes sin alterar.

VI-B1. Resultados: En el siguiente cuadro que se muestra en la Figura 6 VI-B1 se presenta la comparación de imágenes, donde cada fila corresponde a un valor de ϵ y arriba de cada imagen se tiene:
(el número correcto) $\rightarrow$ (el número clasificado obtenido).

Figura 6. Tabla comparativa de imágenes obtenidas bajo diferentes ϵ

Se puede ver en el cuadro anterior que cuando $\epsilon = 0$, no se tiene una modificación en la imagen por lo que la red neuronal clasifica correctamente los números sin problemas. Una vez que el valor de ϵ sube a 0.05 se tienen los primeros errores de la red neuronal aún cuando la imagen no presenta una alteración notoria, esto va cambiando a medida que el valor de ϵ sube hasta llegar a 0.3 donde la imagen presenta un ruido notorio haciendo que se vean borrosos los números. Hay que ser efectivos sin ser obvios en el ataque.

Pero ¿Cuanto afecta este ruido agregado en la precisión de la red neuronal? En el siguiente cuadro se muestra el efecto del ruido agregado en la clasificación de imágenes con respecto a los valores 0.05, 0.1, 0.15, 0.2, 0.25, 0.3 de ϵ

ϵ	Correcto/Total	Precisión
0	9810 / 10000	0.981
0.05	9426 / 10000	0.9426
0.10	8510 / 10000	0.8510
0.15	6826 / 10000	0.6826
0.20	4301 / 10000	0.4301
0.25	2082 / 10000	0.2082
0.30	869 / 10000	0.0869

Con esto tenemos que sin el agregado de ruido, $\epsilon = 0$, la red neuronal clasifica de forma correcta 9 810 imágenes del total de 10 000 (Precisión = 0.9801), en el primer valor de $\epsilon = 0{,}05$ el numero de aciertos baja al solo clasificar de forma correcta 9 426 del total de 10 000 (Precisión = 0.9426), con el aumento del valor de ϵ el número de clasificaciones correctas solo sigue disminuyendo hasta llegar a el valor de $\epsilon = 0{,}3$ donde la red neuronal solo clasifica de forma correcta 869 imágenes del total de 10 000 (Precisión = 0.0869).

A continuación en la Figura 7 VI-B1 mostramos la gráfica de la caída de la precisión dado ϵ. Los datos fueron extraídos de la tabla de la Figura 6 previamente explicada.

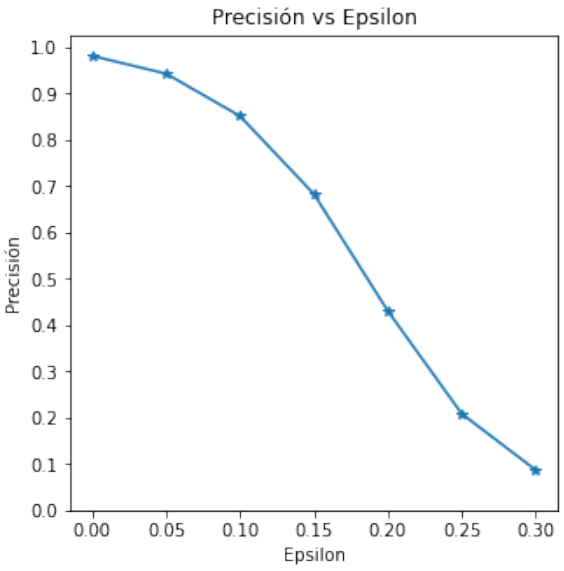

Figura 7. Precisión vs Epsilon(ϵ)

VII. Discusión de Resultados

En el trabajo se implementó el método de Símbolo de Gradiente Rápido (FGSM), durante el trabajo se mostró que una de las formas más simples de atacar una red neuronal es mediante su mismo proceso de aprendizaje, nos aprovechamos del cálculo de gradientes en su función de costo para usarlo en su contra, esto nos brinda el ruido(no un ruido aleatorio) que se debe agregar a la imagen.

Se obtuvo que, a medida que se intensificó el ataque a la red mediante el hiperparámetro ϵ, la imagen se volvió menos reconocible, esto fue debido a que las características de la imagen son maximizadas al usar las gradientes de aprendizaje como ruido agregado. En un entorno real, esto puede ser desventajoso ya que si se quisiera vulnerar una red, lo mejor seria primero engañar a las personas para que piensen que la imagen en cuestión es valida. Una manera de mejorar las salidas del método FGSM seria implementar un limite de ruido agregado debido a que se tiene un $\epsilon = 0{,}15$ la red neuronal disminuye un 30 % de precisión y la imagen sigue siendo legible.

VIII. Conclusiones

- Se lograron generar ejemplos adversarios por medio del FGSM(**fast gradient sign method**) al utilizar diferentes valores del parámetro "ϵ" de la fórmula (4) que varió de 0.05 a 0.3, y creó el *ruido* que al ser procesado por el modelo disminuyó la precisión a medida que el valor de "ϵ" aumentaba, y de este modo se clasificaron las imágenes de manera errónea. Se concluyó que el modelo utilizado con el conjunto de datos MNIST puede engañarse con mucha facilidad y fallar de manera catastrófica.

IX. Trabajos Futuros

En la experimentación se demostró la vulnerabilidad de la red neuronal al conseguir una reducción de la precisión de casi mas de 90 %, sin embargo, esto se logró al volver menos reconocible el número en la imagen, en el futuro se espera poder mejorar el método de perturbación de la imagen para que no se pierda legibilidad y conseguir un ataque **dirigido** y la salida que el atacante desee, con esto se tendrá un ataque que no solo confundirá a las redes neuronales, si no también al ojo humano al no detectar que la imagen que se esta ingresando en la red en verdad está diseñada para obtener una salida diferente.

También se espera poder implementar medidas de protección ante estos ataques por ejemplos adversarios, una forma de identificar cuando una imagen ha sido alterada es mediante la identificación del ruido, una forma de contrarrestar esto es mediante el uso de umbral binario(*binary thresholding*). [7]

Referencias

[1] Ian J. Goodfellow, J. S. . C. S. *Explaining and Harnessing Adversarial Examples*. ICLR, 2015.

[2] Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. Adversarial examples are not bugs, they are features, 2019.

[3] Inkawhich, N. *lenet mnist model*. https://drive.google.com/file/d/1KVOHbHnjCd1L-ookcd7CxDqb7rb8-DSx/view?usp=sharing, 2018.

[4] Shilin Qiu, Qihe Liu, S. Z. . C. W. *Review of Artificial Intelligence Adversarial Attack and Defense Technologies*. MDPI, 2019.

[5] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks, 2014.

[6] Tony Thomas, Athira P. Vijayaraghavan, S. E. *Machine Learning Approaches in Cyber Security Analytics*. Springer, 2020.

[7] Veerapaneni, D. G. . R. *Tricking Neural Networks: Create your own Adversarial Examples*. Machine learning at Berkeley, 2018.

ÍNDICE DE FIGURAS