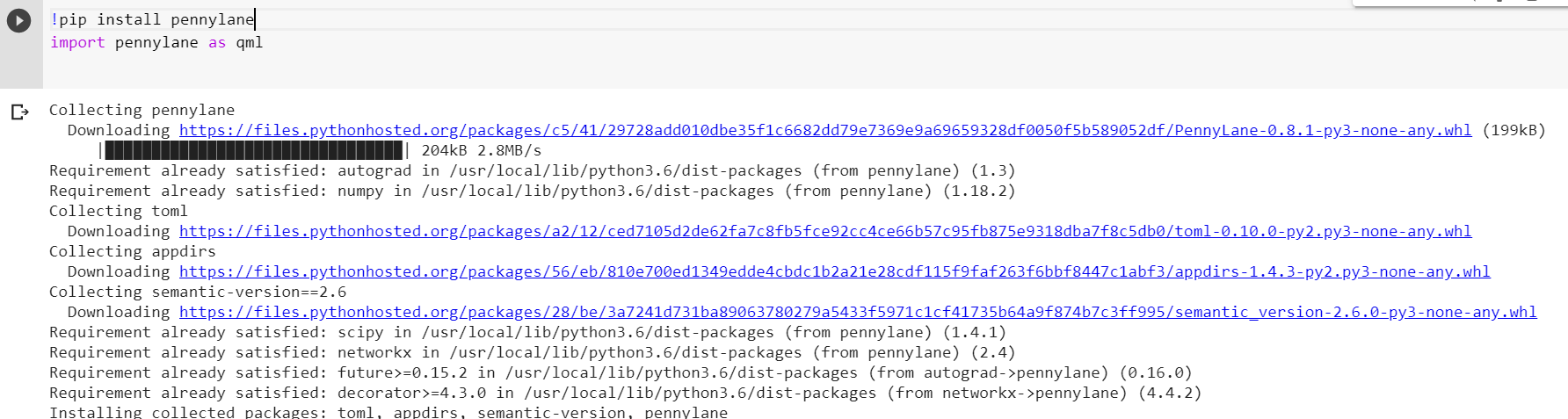
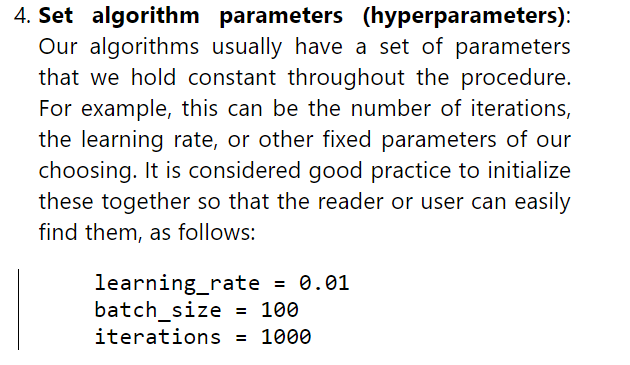
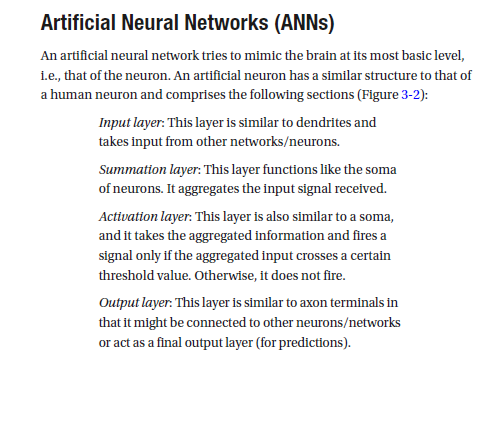
Notas TFG

Instalación de PennyLane en Google Colab









Para implementar este escenario, construimos 3 redes neuronales:

* Alice es el algoritmo de cifrado.
* Bob es el algoritmo de descifrado
* Eva es la atacante

Necesitaremos las siguientes importaciones para nuestra implementación de Keras:

from keras import backend as K

from keras.models import Model

from keras.engine.input\_layer import Input

from keras.layers.core import Activation, Dense

from keras.layers import Flatten, Reshape

from keras.layers.convolutional import Conv1D

from keras.layers.merge import concatenate

from keras.optimizers import Adam, RMSprop

También configuramos los parámetros del modelo. Según el trabajo de investigación, utilizaremos mensajes de 16 bits y claves secretas y textos cifrados de 16 tonos.

# Set up the crypto parameters: message, key, and ciphertext bit lengths

m\_bits = 16

k\_bits = 16

c\_bits = 16

pad = 'same'

# Compute the size of the message space, used later in training

m\_train = 2\*\*(m\_bits) #+ k\_bits)

**Alice / Bob net**

La red de Alice necesitará tener dos vectores de entrada: el mensaje a cifrar y la clave secreta. Estos se concatenan y se unen a una capa densa. La señal luego pasa a través de una secuencia de cuatro capas Conv1D para crear la salida.

ainput0 = Input(shape=(m\_bits,)) #message

ainput1 = Input(shape=(k\_bits,)) #key

ainput = concatenate([ainput0, ainput1], axis=1)

adense1 = Dense(units=(m\_bits + k\_bits))(ainput)

adense1a = Activation('tanh')(adense1)

areshape = Reshape((m\_bits + k\_bits, 1,))(adense1a)

aconv1 = Conv1D(filters=2, kernel\_size=4, strides=1, padding=pad)(areshape)

aconv1a = Activation('tanh')(aconv1)

aconv2 = Conv1D(filters=4, kernel\_size=2, strides=2, padding=pad)(aconv1a)

aconv2a = Activation('tanh')(aconv2)

aconv3 = Conv1D(filters=4, kernel\_size=1, strides=1, padding=pad)(aconv2a)

aconv3a = Activation('tanh')(aconv3)

aconv4 = Conv1D(filters=1, kernel\_size=1, strides=1, padding=pad)(aconv3a)

aconv4a = Activation('sigmoid')(aconv4)

aoutput = Flatten()(aconv4a)

alice = Model([ainput0, ainput1], aoutput, name='alice')

La salida de la capa Densa será bidimensional, con las instancias de mini lotes a lo largo del primer eje. Esto necesita ser reformado (batch\_size, m\_bits + k\_bits, 1)antes de ser unido a una capa Conv1D, introduciendo una tercera dimensión que normalmente correspondería al número de canales (en este caso, solo hay 1).

La arquitectura de la red Bob es idéntica a la de Alice, excepto que input0 representa el texto cifrado en lugar del texto sin formato.

**Neta Eva**

Según las especificaciones en el trabajo de investigación, la red Eve tendrá DOS capas densas que actuarán en las entradas. La idea es darle una mejor oportunidad de descifrar el mensaje, ya que no tiene acceso a la clave secreta, solo ve el texto cifrado.

einput = Input(shape=(c\_bits,)) #ciphertext only

edense1 = Dense(units=(c\_bits + k\_bits))(einput)

edense1a = Activation('tanh')(edense1)

edense2 = Dense(units=(c\_bits + k\_bits))(edense1a)

edense2a = Activation('tanh')(edense2)

ereshape = Reshape((c\_bits + k\_bits, 1,))(edense2a)

econv1 = Conv1D(filters=2, kernel\_size=4, strides=1, padding=pad)(ereshape)

econv1a = Activation('tanh')(econv1)

econv2 = Conv1D(filters=4, kernel\_size=2, strides=2, padding=pad)(econv1a)

econv2a = Activation('tanh')(econv2)

econv3 = Conv1D(filters=4, kernel\_size=1, strides=1, padding=pad)(econv2a)

econv3a = Activation('tanh')(econv3)

econv4 = Conv1D(filters=1, kernel\_size=1, strides=1, padding=pad)(econv3a)

econv4a = Activation('sigmoid')(econv4)

eoutput = Flatten()(econv4a)# Eve's attempt at guessing the plaintext

eve = Model(einput, eoutput, name='eve')

**Pérdida y optimizador**

Para entrenar las redes, primero necesitamos unirlas.

aliceout = alice([ainput0, ainput1])

bobout = bob( [aliceout, binput1] )# bob sees ciphertext AND key

eveout = eve( aliceout )# eve doesn't see the key

La pérdida para Eve es la distancia L1 entre ainput0y eoutput:

eveloss = K.mean( K.sum(K.abs(ainput0 - eveout), axis=-1) )

En lugar de hacer un promedio, se suma la suma de todos los bits en el mensaje; esta es la pérdida *por ejemplo mencionada* en el documento original, y su valor representa el número promedio de bits que Eve adivina incorrectamente. Luego tomamos el promedio en todo el mini lote con K.mean(). El valor mínimo de la pérdida es 0 (Eve adivina todos los bits correctamente), mientras que el máximo es 16 (Eve está equivocada acerca de todos los bits, en cuyo caso volteamos la predicción y recuperamos el mensaje).

La pérdida de Alice-Bob es un poco más complicada. Primero, queremos que Bob descifre con éxito el texto cifrado:

bobloss = K.mean( K.sum(K.abs(ainput0 - bobout), axis=-1) )

También me gustaría que Alice aprenda un esquema de cifrado que Eve no puede romper. En una situación ideal, Eve no debería hacer nada mejor que adivinar al azar, en cuyo caso adivinaría correctamente la mitad de los bits, o m\_bits/2correctamente (correspondiente a un valor de pérdida de 8).

K.square(m\_bits/2 - eveloss)/( (m\_bits//2)\*\*2 )

El término cuadrado asegura que el error aumenta cuando Eve comienza a adivinar algunos de los bits correctamente. Esto obliga a Alice a adaptarse más rápidamente a los atacantes y a cambiar su esquema de cifrado. En general, la pérdida de Alice-Bob será:

abeloss = bobloss + K.square(m\_bits/2 - eveloss)/( (m\_bits//2)\*\*2 )

Según el optimizador, utilizamos RMSprop con una tasa de aprendizaje predeterminada de 0.001:

abeoptim = RMSprop(lr=0.001)

eveoptim = RMSprop(lr=0.001) #default 0.001

Finalmente, creamos dos modelos de entrenamiento con las funciones de pérdida personalizadas definidas anteriormente:

# Build and compile the ABE model, used for training Alice-Bob networks

#

abemodel = Model([ainput0, ainput1, binput1], bobout, name='abemodel')

abemodel.add\_loss(abeloss)

abemodel.compile(optimizer=abeoptim)

# Build and compile the Eve model, used for training Eve net (with Alice frozen)

#

alice.trainable = False

evemodel = Model([ainput0, ainput1], eveout, name='evemodel')

evemodel.add\_loss(eveloss)

evemodel.compile(optimizer=eveoptim)

n\_epochs = 20

**Resultados**

Durante el entrenamiento, realizamos un seguimiento no solo de los valores abelossy eveloss, sino que también evaluamos la capacidad de Bob para descifrar los mensajes enviados por Alice. *La Figura 2* muestra los valores de pérdida a medida que avanzaba el entrenamiento.

|  |
| --- |
| *Figura 2: valores de pérdida durante el entrenamiento* |

Inicialmente, los tres valores de pérdida comienzan en 8, lo que significa que ni Eve ni Bob lo hacen mejor que adivinar al azar. La pérdida de Eve disminuye más rápido hasta cierto punto, después de lo cual Alice puede aprender un esquema de encriptación que Bob puede entender, pero que engaña a Eve. Tenga en cuenta que la pérdida de Eve no vuelve a subir hasta 8, por lo que Eve todavía lo hace un poco mejor que adivinar al azar.

Evaluamos a Bob y Eve en 10,000 mensajes aleatorios (encriptados con claves seleccionadas al azar), y encontramos lo siguiente:

Bob % correct: 89.28 %

Eve % correct: 0.21 %

Bob puede descifrar con éxito un poco menos del 90% de los mensajes que recibe, mientras que *Eve no puede recuperar el mensaje completo en su estado actual* . Sin embargo, si tuviéramos que congelar el modelo Alice-Bob y continuar entrenando a Eve, su desempeño mejoraría.

Es importante darse cuenta de que Eve todavía funciona mejor que adivinar al azar. Para cada bit, una suposición puramente aleatoria tendríaP ( 1 bit correcto ) = 1 2PAG(1 bit correcto)=12

posibilidad de adivinar ese bit correctamente. Bajo el supuesto de conjeturas independientes para cada bit de texto cifrado, la probabilidad de adivinar todo el mensaje correctamente seríaP ( todos los bits correctos ) = ( 1 2) 16 =0.0015%PAG(todos los bits correctos)=(12)dieciséis=0.0015%que es significativamente peor que el rendimiento real de Eve.

Es interesante analizar el desempeño de Eve cuando mantiene congelada a Alice-Bob. *La Figura 3* muestra dos de estos escenarios:

|  |
| --- |
| *Figura 3: rendimiento de Eve para AB congelado al comenzar sin entrenamiento (izquierda) o entrenado (derecha)* |

Al comenzar con redes recién inicializadas y mantener congelada a Alice-Bob, Eve aún puede obtener una ventaja no despreciable. Ella termina con 3 conjeturas incorrectas en promedio. Cuando se evalúa en una muestra de 10,000 textos cifrados, obtenemos

Bob % correct: 0.0 %

Eve % correct: 2.55 %

Sin embargo, al mantener constante la versión entrenada de Alice-Bob, vemos que la ventaja de Eve no comienza a mejorar. La pérdida se mantiene más o menos constante, lo que sugiere que el algoritmo de encriptación aprendido por Alice puede no necesitar ser mejorado continuamente.

