Deep Learning. Tarea 9. AutoEncoders

Sebastián Cadavid-Sánchez - Juan Pablo Herrera Musi 23 de Abril de 2020

Pregunta 1

Diseña un AE que obtenga pérdida menor o igual 0.01 tanto en entrenamiento como validación. Reporta tu arquitectura.

Solución:

La mejor arquitectura AE entrenada obtuvo una pérdida de 0.0152 y 94.71% de precisión para el conjunto de datos de entrenamiento, y 0.0101 y 96.16%, respectivamente, para el conjunto de datos de test. En particular, la fase de entrenamiento asociada a dicho resultado incluyó 100 épocas, utilizando el optimizador "adam". Esta arquitectura es la número 5 reportada en la parte final del notebook anexado.

En el *listing* 1 se reporta el código utilizado para generar la mejor arquitectura y en la figura 2 se presenta la arquitectura asociada.

Listing 1: Arquitectura con mejor desempeño

```
    from tensorflow.keras import regularizers
    # Create model
    in_layer = Input(shape=(x_train.shape[1],))

    encoder = Dense(64, activation='relu')(in_layer)
    encoder = Dense(32,  activation='tanh',
    activity_regularizer=regularizers.l1(10e-5))(encoder)

    latent  = Dense(24,  activation='relu')(encoder)

    decoder = Dense(32, activation='relu',
    activity_regularizer=regularizers.l1(10e-5))(latent)
    decoder = Dense(64, activation='tanh')(decoder)

    out_layer = Dense(x_train.shape[1])(decoder)

    AE = Model(inputs=in_layer, outputs=out_layer)
```

```
Model: "model_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_13 (InputLayer)        [(None, 29)]              0
_________________________________________________________________
dense_76 (Dense)             (None, 64)                1920
_________________________________________________________________
dense_77 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_78 (Dense)             (None, 24)                792
_________________________________________________________________
dense_79 (Dense)             (None, 32)                800
_________________________________________________________________
dense_80 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_81 (Dense)             (None, 29)                1885
=================================================================
Total params: 9,589
Trainable params: 9,589
Non-trainable params: 0
```

Figure 1: Arquitectura de mejor arquitectura con AE.

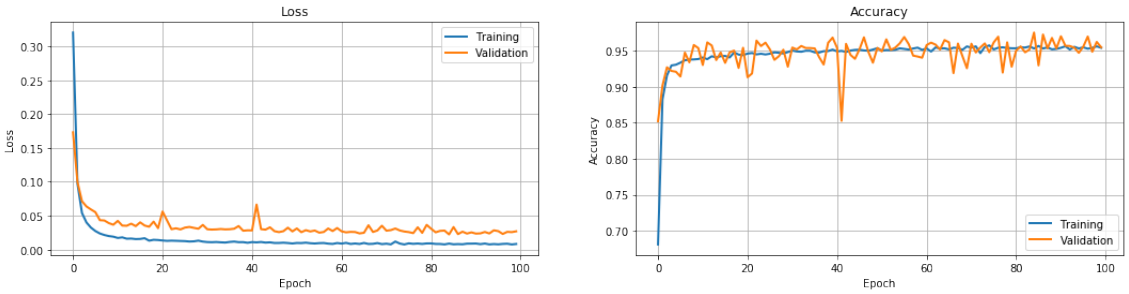

Figure 2: Pérdida y precisión de la Arquitectura de mejor arquitectura con AE para los datos de entrenamiento y prueba.

Como se puede observar en la arquitectura de mejor desempeño, los cambios más relevantes incluidos implicaron la utilización de regularización *l1*, en el *encoder* inicial que incluye una función de activación de tangente hiperbólica, y la que precede al último *decoder*, que posee una función de activación *relu*. De igual forma, la última capa *decoder* incorpora una función de activación de tangente hiperbólica.

Cabe mencionar que, si se observa la figura 2 el proceso de aprendizaje para los datos de prueba es mucho menos suavizado que para el conjunto de datos de entrenamiento. Sin embargo, no hay señales de sobre-entrenamiento, en la medida que las curvas tanto de pérdida como de precisión se comportan de manera similar.

En los incisos posteriores se discutirá sobre algunos de los detalles sobre las arquitecturas de modelos de *autoencoding*.

Pregunta 2

¿Existe alguna relación entre la profundidad del AE y la pérdida final?

Solución:

No existe ninguna relación directa entre la profundidad de autoencoder y la pérdida final. La pérdida final solamente toma en cuenta las diferencias entre el *input* y el *output* y no tiene

dependencia con la profundidad del autoencoder. Por ejemplo, si la función de pérdida es la de error cuadrado medio, tenemos que:

$$loss = ||x - \hat{x}||^2 = ||x - d(z)||^2 = ||x - d(e(x))||^2$$

con:

- x el *input*
- $\hat{x}$ el *output*, que corresponde a la reconstrucción de *input*
- z es la representación de x en el espacio latente
- $d(\cdot)$ es el *decoder*, una función que transforma del espacio latente al espacio original
- $e(\cdot)$ es el *encoder*, una función que transforma del espacio orogonal al espacio latente

Entendemos la profundidad como el número de capas que tiene el encoder antes de alcanzar la capa latente. La relevancia de la profundidad se verá reflejada en las diferentes características del *input* que el encoder es capaz transformar sin perder información. La profundidad nos indicará el número de grados de libertad que se tienen para lograr reducciones de dimensionalidad considerables. Sin embargo no tiene relación directa con la función de pérdida.

Pregunta 3

¿Existe alguna relación entre la profundidad del AE y la separación resultante entre los espacios latentes de los datos normales y anormales?

Solución:

Si existe una relación entre la profundidad del AE y la separación resultante entre los espacios latentes de los dos tipos de datos. El modelo planteado por un autoencoder lleva los datos representados un la dimensión original a una representación en el espacio latente con diferentes dimensiones. Intuitivamente la naturaleza de la arquitectura de red que está entre la capa de entrada y la capa latente determina las transformaciones que se le hacen a los datos de entrada y la representación final que tiene los datos de salida. Dependiendo de los parámetros y número de capas, la red aprenderá características diferentes de los datos y determinará la representación que se le da a los mismos en el espacio latente.

Otra forma de pensar el problema es con el planteamiento matemático del mismo. Podemos simplificar el problema y pensar todo el proceso que realiza el encoder para trasformar el *input* lo podemos modelar como una función $\phi : \chi \to F$, con χ el espacio original y F el espacio latente. Si cambiamos la regla de correspondencia ϕ entonces se generará un *output* diferente aún cuando vive en el mismo espacio. Lo que hacemos al modificar la profundidad es cambiar la función ϕ.

En términos de la red neuronal la función ϕ tendrá la forma de $\phi(\mathbf{x}) = \sigma(\mathbf{W}x + \mathbf{b})$ con $\mathbf{x} \in \chi$ y $\phi(\mathbf{x}) \in F$ para encoders con 1 capa de profundidad. Si aumentamos la profundidad

entonces tendremos composiciones de funciones $\phi(\phi_1(\mathbf{x}))$ y aunque el vector final viva en el mismo espacio tendrá una representación diferente.

En la figura 3 observamos el espacio latente de una red con 5 capas. En la figura 4 observamos el espacio latente de un red con 3 capas. Es posible observar que ambos espacios son muy similares, sin embargo podemos apreciar diferencias sobre todo los puntos que están por las orillas. El motivos de que son diferente se explica en párrafos anteriores, el gran parecido se debe a que están encontrando las mismas características y por tanto representaciones similares.

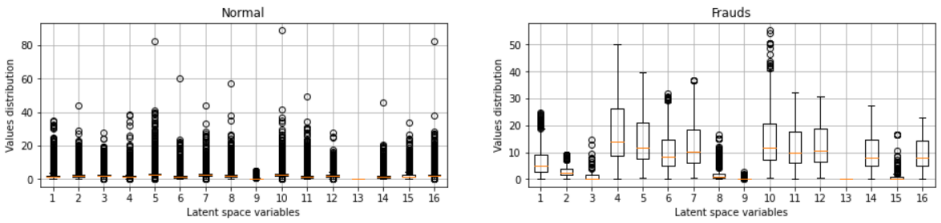

Figure 3: Espacio latente de una red con 5 capas.

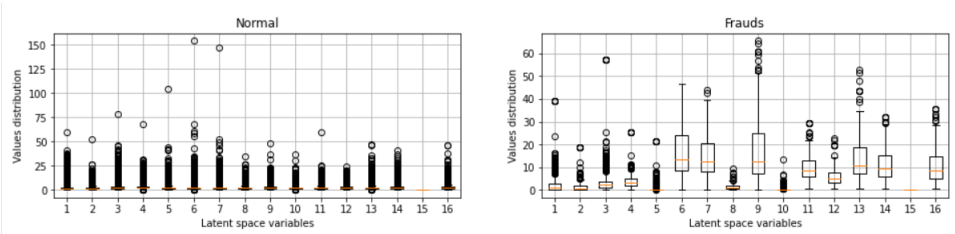

Figure 4: Espacio latente de una red con 3 capas.

Pregunta 4

¿Existe alguna relación entre el número elementos en el espacio latente y la pérdida final?

Solución:

Si, existe una relación indirecta entre el número de elementos en el espacio latente y la pérdida final. Como mencionamos en la pregunta dos, la pérdida final es una función que solamente depende del *input* y el *output*, pero la pérdida contabiliza diferencias entre el input y el output. Por otro lado, el número de elementos en el espacio latente indica la cantidad de información que podremos almacenar de un *input* codificado. Si tenemos muy pocas dimensiones en ese espacio es muy posible que tengamos mucha pérdida de información y al reconstruir la salida tendrá diferencias considerables con el *input*, esto se verá reflejado al evaluar la pérdida.

Una dimensión del espacio latente mayor o igual a la dimensión de entrada permitirá almacenar la información completa del *input*, por lo que esperariamos que se tenga menos pérdida de información y menos pérdida final.

Dimensión Espacio Latente	Pérdida Test	Pérdida Val
4	0.2252	0.2373
8	0.0909	0.0981
16	0.0136	0.0205
32	0.0050	0.0088

Table 1: Pruebas de variación de profundidad.

Para verificar la información se hicieron pruebas con cuatro redes en las que la única variación era el tamaño de la capa latente. La figura 5 muestra una de estas representaciones.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_14 (InputLayer)        [(None, 29)]              0
_________________________________________________________________
dense_94 (Dense)             (None, 128)               3840
_________________________________________________________________
dense_95 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_96 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_97 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_98 (Dense)             (None, 128)               8320
_________________________________________________________________
dense_99 (Dense)             (None, 29)                3741
=================================================================
Total params: 28,349
Trainable params: 28,349
Non-trainable params: 0
```

Figure 5: Arquitectura de red de prueba.

En las otras redes el único hiperparámetro que se modificó fue la dimensión de la capa latente. Se hicieron pruebas con dimensiones 4, 8, 16 y 32. Los resultados se resumen en la tabla 1.

Pregunta 5

¿Existe alguna relación entre el número elementos en el espacio latente y la separación resultante entre los espacios latentes de los datos normales y anormales?

Solución:

Si, existe una relación directa entre el número de elementos en el espacio latente y la separación resultante entre los datos normales y anormales. El número de elementos en el espacio latente determina cuantas separaciones se harán dentro de nuestros datos, por tanto si cambiamos el número de elementos en el espacio latente la separación de los datos cambiará.

Por ejemplo, si utilizamos la misma arquitectura que de 3 capas que se utilizó en la pregunta 3, cuyo espacio latente observamos en la figura 4 pero cambiamos la dimensión de la capa latente de 16 a 8, entonces observamos el espacio latente que se muestra en la figura 6. El

número de divisiones que tiene el espacio latente en este caso es de 8, en el caso anterior era de 16. De modo que la separación de los datos en el espacio latente sí depende de el número de elementos en el mismo.

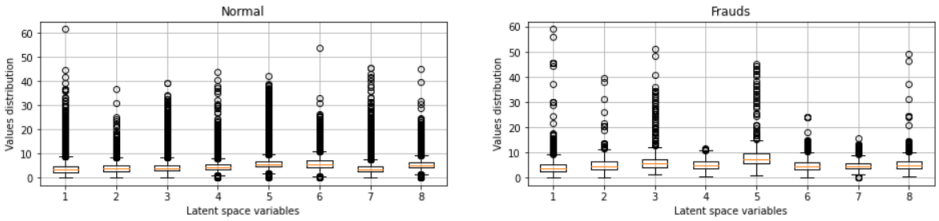

Figure 6: Espacio latente de 8 elementos.

Nota explicativa: Suponemos que la separación de elementos se refiere a su representación en el espacio latente. Si se refiere a transacciones fraudulentas o normales toda la división proviene de que nunca se juntan los datos, por un lado se introducen los fraudulentos y por otro los normales. De forma que nunca etiqueta si es fraudulento o normal.

Pregunta 6

Intenta forzar el espacio latente para que sea ralo. Reporta tu mejor modelo y el desempeño que obtenga, tanto en pérdida como en capacidad de diferenciar datos anormales.

Solución:

En el *listing* 2 se reporta el código utilizado para generar la mejor arquitectura y en la figura 7 se presenta la arquitectura asociada.

Listing 2: Arquitectura con mejor desempeño

```
# Create model
    in_layer = Input(shape=(x_train.shape[1],))

    encoder = Dense(32, activation='relu')(in_layer)
    encoder = Dense(24,  activation='relu')(encoder)
    latent  = Dense(32,activation='relu',activity_regularizer=l1(0.1))(↩
       encoder)
    decoder = Dense(24, activation='relu')(latent)
    decoder = Dense(32, activation='relu')(decoder)
    out_layer = Dense(x_train.shape[1])(decoder)

    AE = Model(inputs=in_layer, outputs=out_layer)
    AE.summary()
```

```
13
14 # Compile
15     AE.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
16
17 # Train it
18     history = AE.fit(x_train, x_train, epochs=30, batch_size=64, ↩
           validation_split=0.2)
```

Layer (type)	Output Shape	Param #
input_14 (InputLayer)	[(None, 29)]	0
dense_94 (Dense)	(None, 128)	3840
dense_95 (Dense)	(None, 64)	8256
dense_96 (Dense)	(None, 32)	2080
dense_97 (Dense)	(None, 64)	2112
dense_98 (Dense)	(None, 128)	8320
dense_99 (Dense)	(None, 29)	3741

Total params: 28,349
Trainable params: 28,349
Non-trainable params: 0

Figure 7: Arquitectura de mejor arquitectura con AE.

En esta arquitectura se obligó a tener un espacio latente ralo. Para lograrlo se utilizó un regularizador LASSO ($l1$) en la capa latente. Se utilizó un factor de regularización de 0.1, mayor al recomendado de 0.01 con el fin de lograr el efecto esperado en el espacio latente. Forzar ese efecto también afectó la pérdida considerablemente, la pérdida final fue de 0.2214 para entrenamiento y de $.4651$ para validación.

En la figura 8 podemos observar que el espacio latente tiene varias posiciones en las que no se representa ningún valor. Sin embargo, la separación entre valores anormales es distinguible. Esto lo podemos observar en la gráfica del error cuadrático medio que se presenta en la figura 9.

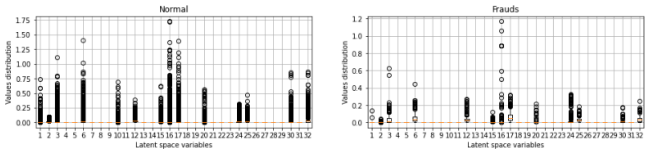

Figure 8: Representación de los datos en el espacio latente para la arquitectura de la figura 7.

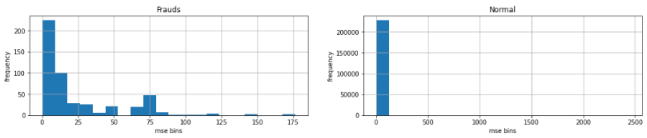

Figure 9: Separación de los datos utilizando la arquitectura de la figura 7.