Inteligencia Artificial Avanzada

Análisis y Reporte sobre el desempeño del modelo

Presentado por:

Felipe Gabriel Yépez Villacreses

A01658002

Fecha de entrega

Domingo 11 de septiembre 2022

Repositorio: https://github.com/FelipeYepez/Heart-Attack-Prediction

Análisis Modelo 1

Mediante el uso de TensorFlow, se replicó la red neuronal implementada usando únicamente Python y numpy, para predecir un ataque cardiaco. Se utilizó un dataset proveniente de Kaggle que se puede encontrar en el siguiente link:

https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset

La arquitectura de la red es la siguiente:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 8)                 112
_________________________________________________________________
dense_19 (Dense)             (None, 2)                 18
=================================================================
Total params: 130
Trainable params: 130
Non-trainable params: 0
_________________________________________________________________
```

Contiene una capa con 13 entradas, una capa oculta con 8 neuronas y una capa de salida con 2 neuronas con los 2 tipos de clasificación: probabilidad de tener un ataque cardiaco y probabilidad de no tenerlo.

```python
# First Model
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(input_dim,)))
model.add(tf.keras.layers.Dense(units=8, activation="relu", kernel_initializer='he_normal'))
model.add(tf.keras.layers.Dense(units=2, activation="softmax", kernel_initializer='he_normal'))

model.compile(
    optimizer=tf.keras.optimizers.SGD(
        learning_rate=0.01,
        momentum=0.0,
        name='SGD'
    ),
    loss="categorical_crossentropy",
    metrics=[tf.keras.metrics.CategoricalAccuracy()]
)
```

La capa oculta tiene ReLU como función de activación mientras que la capa de salida tiene softmax al ser una clasificación lo que se quiere lograr.

Se utilizó un optimizador común que en este caso es gradient descent con momento. Para alcular la función de pérdida se utilizó cross entropy para clasificación. Con la finalidad de medir la precisión del modelo se utilizó una función de precisión que determina si la red clasifica correctamente en la categoría de si el paciente tiene riesgo de tener un ataque cardiaco o no.

Para poder analizar el modelo se lo dividió en 80% train y 20% test. Con la finalidad de obtener resultados repetibles y comparables se utilizó un seed para que la división del conjunto de datos pueda seguir siendo aleatoria y constante a lo largo del tiempo.

```
x_train, x_test, y_train, y_test=train_test_split(X, Y, test_size=0.2, random_state=43)
```

De igual forma, mediante el uso de TensorFlow, al momento de ajustar los parámetros del modelo se separó un 20% de los datos de train para realizar la validación del mismo.

```
history = model.fit(x_train, y_train, epochs=400, validation_split=0.2)
```

Como se puede observar la red neuronal, al igual que la anterior realizada tan solo con Python y numpy, tiene 400 épocas de aprendizaje.

Resultados obtenidos

Con la red neuronal, en la época 400 se obtuvo lo siguiente:

Precisión (Accuracy)		
Train	Validation	Test
0.9067	0.7755	0.7377

```
Epoch 400/400
7/7 [==============================] - 0s 7ms/step - loss: 0.2545 - categorical_accuracy: 0.9067 - val_loss: 0.5196 - val_categorical_accuracy: 0.7755
```

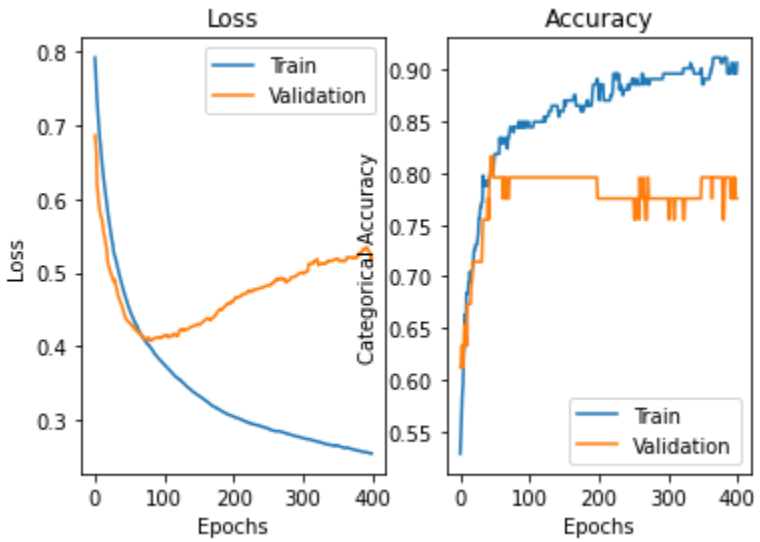
```
evaluate = model.evaluate(x_test, y_test)
print("First Model Categorical Accuracy:", evaluate[1])
```

```
2/2 [==============================] - 0s 5ms/step - loss: 0.5834 - categorical_accuracy: 0.7377
First Model Categorical Accuracy: 0.7377049326896667
```

Con los anteriores datos se puede observar que el modelo está siendo overfitted o sobre-entrenado dado que en el conjunto de entrenamiento está obteniendo un resultado asombroso comparado al conjunto de validación y pruebas.

De lo anterior, se sabe que el modelo tiene alta varianza al estar sobre-entrenado.

De igual forma se obtuvieron las siguientes gráficas durante el entrenamiento del modelo para el conjunto de test y validation.



En la gráfica de pérdida se puede observar cómo el conjunto de validación comienza a aumentar su valor mientras el de entrenamiento disminuye. Se observa cómo antes de la época 100 se empieza a sobre-entrenar el modelo.

De igual forma la precisión del conjunto de validación empieza a disminuir a medida que el modelo se especializa en el conjunto de entrenamiento.

El modelo cuenta con alta varianza y bajo sesgo por los anteriores hechos.

Mejora del modelo

Dado que el modelo cuenta con alta arianza se implementaron distintas técnicas de regularización.

La primera y más clara fue implementar la técnica de early stopping, es decir dejar de entrenar en un número de época menor con la finalidad de que el modelo no se optimice tanto para el conjunto de prueba.

Para lograr esto se fue disminuyendo el número de épocas total progresivamente con el resto de cambios al modelo hasta que el costo de validación no empiece a aumentar.

De igual forma se implementó un método de optimización para entrenar el conjunto de entrenamiento en conjuntos más pequeños llamados mini batches.

```
batch_size = 64
history2 = model2.fit(x_train, y_train,
    epochs=150,
    steps_per_epoch=len(x_train)/batch_size,
    verbose=2,
    validation_split=0.2
)
```

La segunda técnica de regularización para reducir la varianza que se implementó fue agregar dropout a las capas de la red. Lo que hace esta técnica es apagar temporalmente ciertas neuronas con la finalidad de que la red no se vuelva completamente dependiente en un grupo reducido de neuronas sino que todas contribuyan a la obtención del resultado final. Se reduce la sobre especialización de la red reduciendo la varianza.

De igual forma se probaron distintas arquitecturas obteniendo la siguiente con una capa adicional y el uso de diferentes funciones de activación para las capas ocultas.

```
model2 = tf.keras.Sequential()
model2.add(tf.keras.Input(shape=(input_dim,)))
model2.add(tf.keras.layers.Dense(units=8, activation="tanh", kernel_initializer=tf.keras.initializers.HeUniform()))
model2.add(tf.keras.layers.Dropout(0.25))
model2.add(tf.keras.layers.Dense(units=4, activation="tanh", kernel_initializer=tf.keras.initializers.HeUniform()))
model2.add(tf.keras.layers.Dropout(0.25))
model2.add(tf.keras.layers.Dense(units=2, activation="softmax", kernel_initializer=tf.keras.initializers.HeUniform()))
```

```
model2.compile(
    optimizer="RMSprop",
    loss="categorical_crossentropy",
    metrics=[tf.keras.metrics.CategoricalAccuracy()]
)
```

Agregar más capas aumenta la complejidad de la red y por lo tanto reduce el sesgo, aumentando la varianza. A pesar de posiblemente aumentar la varianza, se utilizaron las técnicas de regularización anteriores para prevenir aumentar la varianza.

Resultados obtenidos con mejoras

Precisión (Accuracy)		
Train	Validation	Test
0.8290	0.8571	0.8525

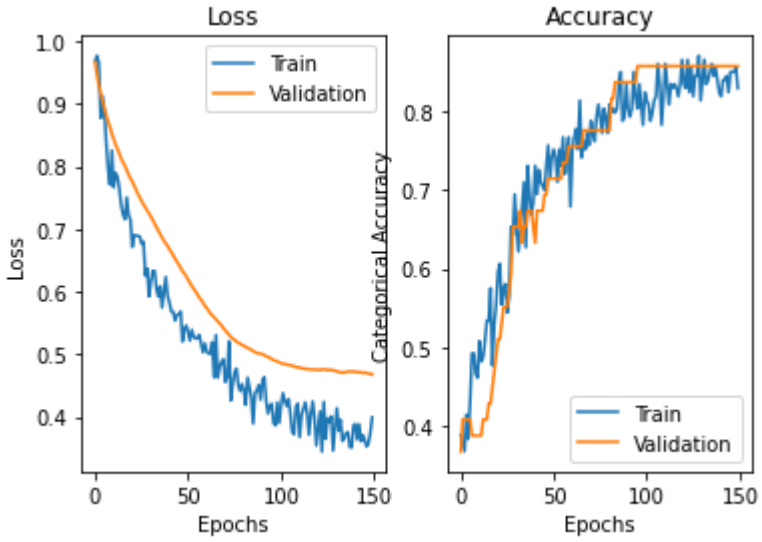
```
Epoch 150/150
3/3 - 0s - loss: 0.3994 - categorical_accuracy: 0.8290 - val_loss: 0.4682 - val_categorical_accuracy: 0.8571
```

```
evaluate2 = model2.evaluate(x_test, y_test)
print("Improved Model Categorical Test Accuracy:", evaluate2[1])
```

```
2/2 [==============================] - 0s 5ms/step - loss: 0.3787 - categorical_accuracy: 0.8525
Improved Model Categorical Test Accuracy: 0.8524590134620667
```

Se puede observar cómo la precisión del conjunto de validación y pruebas aumento mientras se redujo la precisión del conjunto de entrenamiento. Se está obteniendo varianza baja y un sesgo entre normal y alto dado que no se obtienen resultados tan precisos como se quisiera.

Con los cambios se obtuvieron las siguientes gráficas durante el entrenamiento del modelo para el conjunto de test y validation.



En la gráfica de pérdida se puede observar cómo el conjunto de validación ya no empieza a aumentar su valor. Ya no se observa varianza alta y por lo tanto ya no se está sobre-entrenando el modelo.

Se puede observar cómo el costo del conjunto de validación empieza a ser constante, razón por la cuál se puede asumir que se entrenó lo suficiente el conjunto antes de que el mismo tenga mayor pérdida.

La precisión del conjunto de validación alcanza su punto máximo y se vuelve constante sin disminuir por lo que se comprueba el anterior supuesto.

El modelo cuenta con baja varianza y sesgo entre normal y alto por los anteriores hechos, sin embargo no se pudo mejorar el mismo para disminuir el sesgo. Esto puede deberse a la calidad del conjunto de datos al contar con relativamente pocos registros.