KNN

May 10, 2023

1 Actividad

1.1 Descripción

Desde que se secuenció el primer genoma humano en 2001, la caracterización genómica integral de los tumores se ha convertido en uno de los objetivos principales de los investigadores en cáncer. El Consorcio Pan-Cancer Analysis of Whole Genomes supuso un paso decisivo con el atlas del genoma del cáncer completo. La implicación clínica de Pan-Cancer se enmarca en la medicina personalizada. Esta recopilación de datos es parte del conjunto de datos PANCAN de RNA-Seq, es un subconjunto de expresiones génicas de pacientes que tienen diferentes tipos de tumores: Breast invasive carcinoma (BRCA), Kidney renal clear cell carcinoma (KIRC), Colon adenocarcinoma (COAD), Lung adenocarcinoma (LUAD) y Prostate adenocarcinoma (PRAD). Los datos para esta actividad se encuentran en dos ficheros en el zip TCGA-PANCAN-HiSeq-801x20531. El fichero expression.csv y el fichero labels.csv. El fichero expression.csv con las expresiones de genes, tiene 801 filas (muestras) y 20532 columnas. La primera columna es el codigo de la muestra y el resto de columnas indican los genes. Dado su tamaño, el fichero puede requerir unos segundos para ser cargado. El fichero labels.csv con las etiquetas del tipo de tumor, tiene 801 filas (muestras) y dos columnas, la primera con el codigo de muestra y la segunda con la etiqueta de la clase de tumor. El objetivo planteado es la implementación y evaluación de una red neuronal basada en capas densas (fc) para la clasificación de la clase de tumor a partir del transcriptoma en celulas tumorales.

1.2 Enunciado

Se pide:

Cargar los datos (expression.csv y labels.csv)

[3]:
```
import pandas as pd

expression_df= pd.read_csv('/content/drive/MyDrive/Machine Learning/PEC2/
 ↪TCGA-PANCAN-HiSeq-801x20531/expression.csv', index_col=0)
labels_df= pd.read_csv('/content/drive/MyDrive/Machine Learning/PEC2/
 ↪TCGA-PANCAN-HiSeq-801x20531/labels.csv', index_col=0)

expression_df.head()
```

[3]:
```
          gene_0    gene_1    gene_2    gene_3     gene_4  gene_5    gene_6  \
X
sample_0     0.0  2.017209  3.265527  5.478487  10.431999       0  7.175175
```

```
sample_1     0.0  0.592732  1.588421  7.586157   9.623011       0  6.816049
sample_2     0.0  3.511759  4.327199  6.881787   9.870730       0  6.972130
sample_3     0.0  3.663618  4.507649  6.659068  10.196184       0  7.843375
sample_4     0.0  2.655741  2.821547  6.539454   9.738265       0  6.566967

            gene_7  gene_8  gene_9  …  gene_20521  gene_20522  gene_20523  \
X                                   …
sample_0  0.591871     0.0     0.0  …    4.926711    8.210257    9.723516
sample_1  0.000000     0.0     0.0  …    4.593372    7.323865    9.740931
sample_2  0.452595     0.0     0.0  …    5.125213    8.127123   10.908640
sample_3  0.434882     0.0     0.0  …    6.076566    8.792959   10.141520
sample_4  0.360982     0.0     0.0  …    5.996032    8.891425   10.373790

          gene_20524  gene_20525  gene_20526  gene_20527  gene_20528  \
X
sample_0    7.220030    9.119813   12.003135    9.650743    8.921326
sample_1    6.256586    8.381612   12.674552   10.517059    9.397854
sample_2    5.401607    9.911597    9.045255    9.788359   10.090470
sample_3    8.942805    9.601208   11.392682    9.694814    9.684365
sample_4    7.181162    9.846910   11.922439    9.217749    9.461191

          gene_20529  gene_20530
X
sample_0    5.286759         0.0
sample_1    2.094168         0.0
sample_2    1.683023         0.0
sample_3    3.292001         0.0
sample_4    5.110372         0.0

[5 rows x 20531 columns]
```

[ ]:
```
labels_df.head()
```

[ ]:
```
          Class
X
sample_0  PRAD
sample_1  LUAD
sample_2  PRAD
sample_3  PRAD
sample_4  BRCA
```

Seleccionar los genes con mayor variabilidad. Como umbral de corte se utilizará el percentil 80 de la distribución de las varianzas de las expresiones de los genes. Nota: Con dicho filtrado deberian quedar seleccionados 4102 genes

[4]:
```
import numpy as np
```

```
# Calculo de la varianza de cada gen
variances= expression_df.var(axis=0)
print(variances)

# Seleccionar los genes con varianza por encima del percentil 80
threshold = variances.quantile(0.8002)
selected_genes = variances[variances >= threshold].index.tolist()


# Crear un nuevo dataframe con los genes seleccionados
df_genes = expression_df[selected_genes]

# Imprimir el número de genes seleccionados
print(f'Se han seleccionado {len(selected_genes)} genes.')
```

```
gene_0        0.018728
gene_1        1.441987
gene_2        1.135506
gene_3        0.408089
gene_4        0.256580
                ...
gene_20526    0.449397
gene_20527    0.337060
gene_20528    0.317926
gene_20529    4.300892
gene_20530    0.132881
Length: 20531, dtype: float64
Se han seleccionado 4102 genes.
```

Normalizar las expresiones con la transformación minmax

[5]:
```
from sklearn.preprocessing import MinMaxScaler

# Creamos un objeto MinMaxScaler
scaler = MinMaxScaler()

# Escalamos los datos de expresión
expression_scaled = scaler.fit_transform(df_genes.values)

# Creamos un nuevo DataFrame con los datos escalados
expression_scaled_df = pd.DataFrame(expression_scaled, columns=df_genes.columns)

print(expression_scaled_df.shape)
```

```
(801, 4102)
```

Separar los datos en train (2/3) y test (1/3)

[6]:
```
from sklearn.model_selection import train_test_split

# Aplicar onehot encoding al dataframe de etiquetas
labels_encoded = pd.get_dummies(labels_df['Class'])

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(expression_scaled_df,␣
 ↪labels_encoded, test_size=0.33, train_size= 0.66, random_state=1)

print("Dimensiones del conjunto de entrenamiento:", X_train.shape)
print("Dimensiones del conjunto de prueba:", X_test.shape)
```

```
Dimensiones del conjunto de entrenamiento: (528, 4102)
Dimensiones del conjunto de prueba: (265, 4102)
```

Definir el modelo 1, que consiste en una red neuronal con una capa oculta densa de 100 nodos, con activación relu. Añadir un 30% de dropout. Proporcionar el summary del modelo y justificar el total de parámetros de cada capa

[7]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras import optimizers

model1 = Sequential([
    Dense(100, input_shape=(len(X_train.columns),), activation='relu'),
    Dropout(0.3),
    Dense(5, activation='softmax')
])

model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               410300

 dropout (Dropout)           (None, 100)               0

 dense_1 (Dense)             (None, 5)                 505

=================================================================
Total params: 410,805
Trainable params: 410,805
Non-trainable params: 0
_________________________________________________________________
```

- Capa Dense: esta capa tiene 100 nodos, y como entrada recibe un tensor de dimensiones correspondientes a las 4107 características de entrada de cada instancia en el conjunto de

datos. La cantidad total de parámetros en esta capa se calcula como (4107 características de entrada * 100 nodos) + 100 términos de sesgo, lo que resulta en 410,805 parámetros.

- Capa Dropout: esta capa simplemente aplica una operación de "apagado" (o eliminación) aleatoria del 30% de las salidas de la capa anterior, lo que no requiere ningún parámetro adicional.

Ajustar el modelo 1 con un 20% de validación, mostrando la curva de aprendizaje de entrenamiento y validación con 10 epocas

[14]:
```
model1.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

batch_size = 64
epochs = 10
val_split = 0.2

print("Fit model on training data...")
# Ajustar el modelo con las nuevas etiquetas categóricas
history = model1.fit(X_train, y_train, epochs=10, verbose=2, validation_split=0.
↪2)
```

```
Fit model on training data…
Epoch 1/10
14/14 - 1s - loss: 0.0046 - accuracy: 0.9976 - val_loss: 2.5786e-04 -
val_accuracy: 1.0000 - 1s/epoch - 80ms/step
Epoch 2/10
14/14 - 0s - loss: 0.0150 - accuracy: 0.9929 - val_loss: 0.0030 - val_accuracy:
1.0000 - 81ms/epoch - 6ms/step
Epoch 3/10
14/14 - 0s - loss: 0.0253 - accuracy: 0.9929 - val_loss: 1.6586e-04 -
val_accuracy: 1.0000 - 83ms/epoch - 6ms/step
Epoch 4/10
14/14 - 0s - loss: 0.0100 - accuracy: 0.9976 - val_loss: 1.6433e-04 -
val_accuracy: 1.0000 - 85ms/epoch - 6ms/step
Epoch 5/10
14/14 - 0s - loss: 0.0019 - accuracy: 1.0000 - val_loss: 3.6139e-05 -
val_accuracy: 1.0000 - 90ms/epoch - 6ms/step
Epoch 6/10
14/14 - 0s - loss: 0.0053 - accuracy: 0.9976 - val_loss: 1.5932e-05 -
val_accuracy: 1.0000 - 87ms/epoch - 6ms/step
Epoch 7/10
14/14 - 0s - loss: 4.9703e-04 - accuracy: 1.0000 - val_loss: 4.5743e-05 -
val_accuracy: 1.0000 - 73ms/epoch - 5ms/step
Epoch 8/10
14/14 - 0s - loss: 0.0016 - accuracy: 1.0000 - val_loss: 3.6651e-05 -
```

```
val_accuracy: 1.0000 - 77ms/epoch - 5ms/step
Epoch 9/10
14/14 - 0s - loss: 2.1205e-04 - accuracy: 1.0000 - val_loss: 3.4496e-05 -
val_accuracy: 1.0000 - 70ms/epoch - 5ms/step
Epoch 10/10
14/14 - 0s - loss: 3.0643e-04 - accuracy: 1.0000 - val_loss: 2.8100e-05 -
val_accuracy: 1.0000 - 84ms/epoch - 6ms/step
```

7- Obtener la tabla de clasificación erronea del test. Y las métricas usuales de evaluación

Para obtener la tabla de clasificación errónea, primero necesitamos hacer predicciones con el modelo entrenado y luego comparar esas predicciones con las etiquetas reales para ver cuáles fueron clasificadas incorrectamente. Para ello usamos predict()

[15]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
 ↪recall_score, f1_score

# Realizar la predicción sobre el conjunto de prueba
y_pred = model1.predict(X_test)

# Obtener las etiquetas predichas
y_pred_classes = np.argmax(y_pred, axis=1)

# Obtener las etiquetas reales
y_test_classes = np.argmax(y_test.values, axis=1)

# Obtener la matriz de confusión
conf_matrix = confusion_matrix(y_test_classes, y_pred_classes)

# Calcular las métricas de precisión, sensibilidad y especificidad para cada␣
 ↪clase
precision = precision_score(y_test_classes, y_pred_classes, average=None,␣
 ↪zero_division=0)
recall = recall_score(y_test_classes, y_pred_classes, average=None,␣
 ↪zero_division=0)
accuracy = accuracy_score(y_test_classes, y_pred_classes)
f1 = f1_score(y_test_classes, y_pred_classes, average=None, zero_division=0)
specificity = recall_score(y_test_classes, y_pred_classes, pos_label=0,␣
 ↪average=None, zero_division=0)

# Construir la tabla de clasificación errónea
table = pd.DataFrame({'Precision': precision, 'Sensibilidad': recall,␣
 ↪'Especificidad': specificity, 'F1-score': f1},
                     index=labels_encoded.columns)
table
```

```
9/9 [==============================] - 0s 3ms/step

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1396:
UserWarning: Note that pos_label (set to 0) is ignored when average != 'binary'
(got None). You may use labels=[pos_label] to specify a single positive class.
  warnings.warn(
```

[15]:

	Precision	Sensibilidad	Especificidad	F1-score
BRCA	0.990099	1.000000	1.000000	0.995025
COAD	1.000000	1.000000	1.000000	1.000000
KIRC	1.000000	1.000000	1.000000	1.000000
LUAD	1.000000	0.979592	0.979592	0.989691
PRAD	1.000000	1.000000	1.000000	1.000000

Los valores de precisión, sensibilidad, especificidad y F1-score son muy bajos para algunas clases, lo que indica que el modelo puede no estar funcionando correctamente para esas clases.

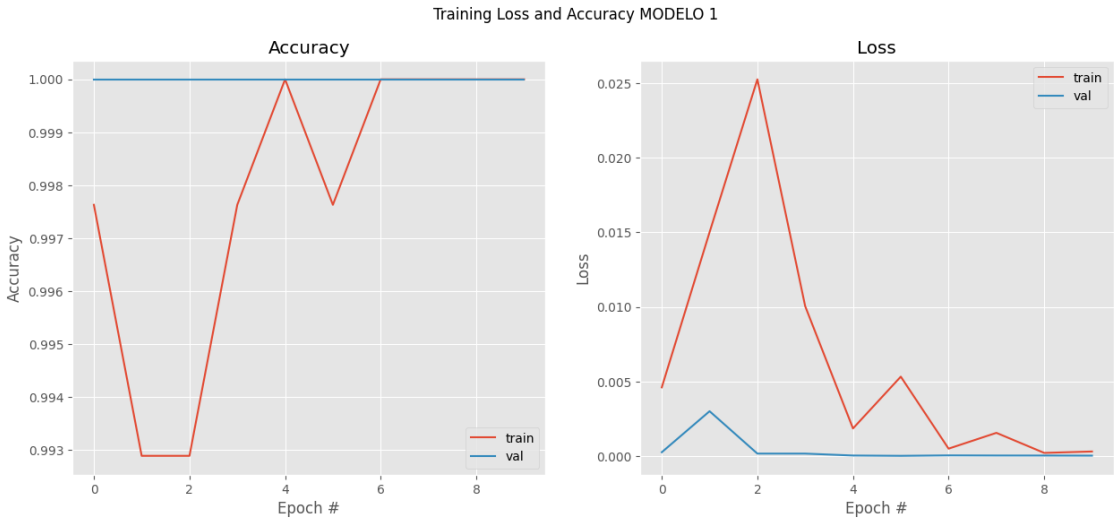
[23]:
```python
# Grafico
import matplotlib.pyplot as plt
def plot_prediction(n_epochs, mfit,title):
  N = n_epochs
  plt.style.use("ggplot")
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,6))
  fig.suptitle(title)

  ax1.plot(np.arange(0, N), mfit.history['accuracy'], label="train")
  ax1.plot(np.arange(0, N), mfit.history["val_accuracy"], label="val")
  ax1.set_title("Accuracy")
  ax1.set_xlabel("Epoch #")
  ax1.set_ylabel("Accuracy")
  ax1.legend(loc="lower right")

  ax2.plot(np.arange(0, N), mfit.history["loss"], label="train")
  ax2.plot(np.arange(0, N), mfit.history["val_loss"], label="val")
  ax2.set_title("Loss")
  ax2.set_xlabel("Epoch #")
  ax2.set_ylabel("Loss")
  ax2.legend(loc="upper right")

  plt.show()

plot_prediction(10, history,'Training Loss and Accuracy MODELO 1')
```



8- Definir el modelo 2, que consiste en una red neuronal con dos capas ocultas densas de 100 nodos y 10 nodos, con activación relu. Añadir un 30% de dropout en ambas capas. Proporcionar el summary del modelo y justificar el total de parámetros de cada capa

[9]:
```
# Definir modelo 2
model2 = Sequential([
    Dense(100, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(10, activation='relu'),
    Dropout(0.3),
    Dense(3, activation='softmax')
])

# Mostrar summary del modelo
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_2 (Dense)              (None, 100)               410300

 dropout_1 (Dropout)          (None, 100)               0

 dense_3 (Dense)              (None, 10)                1010

 dropout_2 (Dropout)          (None, 10)                0

 dense_4 (Dense)              (None, 3)                 33
```

```
=================================================================
Total params: 411,343
Trainable params: 411,343
Non-trainable params: 0
_________________________________________________________________
```

- Capa Dense 1: esta capa tiene 100 nodos, y como entrada recibe un tensor de dimensiones (None, 4107) correspondientes a las 4107 características de entrada de cada instancia en el conjunto de datos. La cantidad total de parámetros en esta capa se calcula como (4107 características de entrada * 100 nodos) + 100 términos de sesgo, lo que resulta en 410,800 parámetros.
- Capa Dropout 1: esta capa simplemente aplica una operación de "apagado" (o eliminación) aleatoria del 30% de las salidas de la capa anterior, lo que no requiere ningún parámetro adicional.
- Capa Dense 2: esta capa tiene 10 nodos, y como entrada recibe el tensor de salida de la capa anterior con dimensiones (None, 100). La cantidad total de parámetros en esta capa se calcula como (100 entradas * 10 nodos) + 10 términos de sesgo, lo que resulta en 1,010 parámetros.
- Capa Dropout 2: de manera similar a la Capa Dropout 1, esta capa no requiere ningún parámetro adicional.
- Capa Dense 3: esta capa tiene 3 nodos, correspondientes a las 3 clases de salida en el conjunto de datos. Como entrada recibe el tensor de salida de la capa anterior con dimensiones (None, 10). La cantidad total de parámetros en esta capa se calcula como (10 entradas * 3 nodos) + 3 términos de sesgo, lo que resulta en 33 parámetros.

9- Ajustar el modelo 2 con un 20% de validación, mostrando la curva de aprendizaje de entrenamiento y validación con 10 epocas

[24]:
```
model2.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

batch_size = 64
epochs = 10
val_split = 0.2

print("Fit model on training data...")
# Ajustar el modelo con las nuevas etiquetas categóricas
history = model2.fit(X_train, y_train, epochs=10, verbose=False,␣
 ↪validation_split=0.2)
```

```
Fit model on training data…
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-24-357e06c48fc1> in <cell line: 13>()
```

```
    11 print("Fit model on training data…")
    12 # Ajustar el modelo con las nuevas etiquetas categóricas
---> 13 history = model2.fit(X_train, y_train, epochs=10, verbose=False,␣
 ↪validation_split=0.2)

/usr/local/lib/python3.10/dist-packages/keras/utils/traceback_utils.py in␣
 ↪error_handler(*args, **kwargs)
    68             # To get the full stack trace, call:
    69             # `tf.debugging.disable_traceback_filtering()`
---> 70             raise e.with_traceback(filtered_tb) from None
    71         finally:
    72             del filtered_tb

/usr/local/lib/python3.10/dist-packages/keras/engine/training.py in␣
 ↪tf__train_function(iterator)
    13                 try:
    14                     do_return = True
---> 15                     retval_ = ag__.converted_call(ag__.
 ↪ld(step_function), (ag__.ld(self), ag__.ld(iterator)), None, fscope)
    16                 except:
    17                     do_return = False

ValueError: in user code:

    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py",␣
 ↪line 1284, in train_function  *
        return step_function(self, iterator)
    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py",␣
 ↪line 1268, in step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py",␣
 ↪line 1249, in run_step  **
        outputs = model.train_step(data)
    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py",␣
 ↪line 1051, in train_step
        loss = self.compute_loss(x, y, y_pred, sample_weight)
    File "/usr/local/lib/python3.10/dist-packages/keras/engine/training.py",␣
 ↪line 1109, in compute_loss
        return self.compiled_loss(
    File "/usr/local/lib/python3.10/dist-packages/keras/engine/compile_utils.
 ↪py", line 265, in __call__
        loss_value = loss_obj(y_t, y_p, sample_weight=sw)
    File "/usr/local/lib/python3.10/dist-packages/keras/losses.py", line 142, i␣
 ↪__call__
        losses = call_fn(y_true, y_pred)
    File "/usr/local/lib/python3.10/dist-packages/keras/losses.py", line 268, i␣
 ↪call  **
        return ag_fn(y_true, y_pred, **self._fn_kwargs)
```

```
  File "/usr/local/lib/python3.10/dist-packages/keras/losses.py", line 1984,␣
↪in categorical_crossentropy
    return backend.categorical_crossentropy(
  File "/usr/local/lib/python3.10/dist-packages/keras/backend.py", line 5559,
↪in categorical_crossentropy
    target.shape.assert_is_compatible_with(output.shape)

ValueError: Shapes (None, 5) and (None, 3) are incompatible
```

10- Comparar en test, mediante las métricas de evaluación, los dos modelos

[ ]:
```
# Realizar la predicción sobre el conjunto de prueba
y_pred = model1.predict(X_test)

# Obtener las etiquetas predichas
y_pred_classes = np.argmax(y_pred, axis=1)

# Obtener las etiquetas reales
y_test_classes = np.argmax(y_test.values, axis=1)

# Obtener la matriz de confusión
conf_matrix = confusion_matrix(y_test_classes, y_pred_classes)

# Calcular las métricas de precisión, sensibilidad y especificidad para cada␣
↪clase
precision = precision_score(y_test_classes, y_pred_classes, average=None,␣
↪zero_division=0)
recall = recall_score(y_test_classes, y_pred_classes, average=None,␣
↪zero_division=0)
accuracy = accuracy_score(y_test_classes, y_pred_classes)
f1 = f1_score(y_test_classes, y_pred_classes, average=None, zero_division=0)
specificity = recall_score(y_test_classes, y_pred_classes, pos_label=0,␣
↪average=None, zero_division=0)

# Construir la tabla de clasificación errónea
table2 = pd.DataFrame({'Precision': precision, 'Sensibilidad': recall,␣
↪'Especificidad': specificity, 'F1-score': f1},
                      index=labels_encoded.columns)
table2
```

1.3 Debate

3. ¿Cuál es la función ReLU? y ¿Cuál es su utilidad?

La función Rectified Linear Unit (ReLU) es una función de activación, es decir, una función que nos permite encontrar relaciones no lineales en los datos en un modelo, más extendida en inteligencia

artificial. És de las más simples: si la entrada es positiva, la salida es igual a la entrada. Por el contrario, si la entrada es negativa, la salida es 0. Dicho de otra forma, es como si ReLU 'apagara' las neuronas que no estan activas, lo que ayuda a mejorar la eficiencia y la capacidad de la generazición nueronal.

Su utilidad reside en su simplicidad y capacidad de generalización, lo que significa que se pueden econocer patrones más complejos en los datos de entrada y producir mejores resultados en tareas de clasificación y reconocimiento de patrones(Lecun et al., 2015). También es interesante destacar que ReLU puede ayudar a prevenir el problema de desvanecimiento de gradientes, que es común en redes neuronales más profundas. Esto se debe a que ReLU solo anula los valores negativos, lo que evita que los gradientes se desvanezcan a medida que se propagan a través de la red (Chollet, 2021)

Por supuesto, como todas las funciones de activación, ReLU no es perfecta y tiene algunas limitaciones, como la posibilidad de que algunas neuronas se "apaguen" por completo y no contribuyan en absoluto a la salida de la red. Sin embargo, en general, ReLU ha demostrado ser una función de activación muy efectiva en una amplia variedad de tareas de aprendizaje profundo, y se ha utilizado con éxito en muchas aplicaciones prácticas, desde la clasificación de imágenes hasta el procesamiento del lenguaje natural.

Referencias:

- Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
- Chollet, F. (2021). Deep Learning with Python. Manning Publications.

2. ¿Cuál es la utilidad del "stochastic gradient learning"?

El SGD o "Stochastic Gradient Learning" es una técnica de optimización en el entrenamiento de redes neuronales.

Al contrario de la optimización de descenso de gradiente clásico, que calcula el gradiente de la función de pérdida utilizando todo el conjunto de datos de entrenamiento; SGD calcula el gradiente utilizando solo un subconjunto aleatorio de los datos de entrenamiento en cada iteración. Eso le confiere varias ventajas e inconvenientes:

Como ventajas; - En primer lugar, permite entrenar redes neuronales profundas en grandes conjuntos de datos con una cantidad limitada de memoria, ya que solo se necesita cargar una pequeña cantidad de datos en la memoria en cada iteración. - Además, el uso de subconjuntos aleatorios de datos de entrenamiento en cada iteración puede ayudar a evitar que el modelo se ajuste demasiado a los datos de entrenamiento y, por lo tanto, mejorar la capacidad de generalización del modelo. - Es más eficiente, ya que los cálculos del gradiente se realizan en subconjuntos más pequeños de los datos. Esto puede acelerar significativamente el proceso de entrenamiento y permitir entrenar modelos más grandes y complejos en un tiempo razonable.

Como desventajas; - El proceso de entrenamiento puede ser menos estable y converger más lentamente que con el descenso de gradiente clásico, debido a la variabilidad introducida por el uso de subconjuntos aleatorios de datos de entrenamiento. - La elección de la tasa de aprendizaje y otros hiperparámetros puede ser más difícil con SGD.

Referencias:

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444. Chollet, F. (2021). Deep Learning with Python. Manning Publications.

4. En la práctica, ¿Son un problema los mínimos locales en el entrenamiento de grandes Neural Networks?

Los mínimos locales, aunque pueden ralentizar el progreso de entrenamiento, no necesariamente son un obstáculo insuperable. En algunos casos, el optimizador puede quedar atrapado en un mínimo local y no poder escapar de él para encontrar un mejor mínimo global. De tal forma que, creiend haver llegado a un òptimo, realmente es un subòptimo. Por suerte, existen técnicas avanzadas de optimización que pueden ayudar a superar estos problemas y lograr un mejor rendimiento del modelo (Chollet, F. (2021). Deep Learning with Python)

8. ¿Qué son las Memory Neural Networks? Descripción. Propósito y Aplicaciones.

La idea detrás de las MNN (Memory Neural Networks) es inspirada en cómo funciona la memoria humana, en la que se almacenan experiencias pasadas y conocimientos previos para ayudar en la toma de decisiones futuras. En lugar de simplemente procesar los datos de entrada de manera secuencial, las MNN almacenan una representación de los datos previos y los usan como contexto para la entrada actual. Esto les permite realizar tareas más complejas, como la comprensión del lenguaje natural y la inferencia en razonamiento humano. Comparadas con las redes neuronales convencionales, las MNN pueden ser más complejas de entrenar y requerir más recursos computacionales. Sin embargo, su capacidad para retener y utilizar información a largo plazo las convierte en una herramienta valiosa para aplicaciones de procesamiento de lenguaje natural y otras aplicaciones que involucran datos secuenciales. Las MNN se utilizan en una amplia variedad de aplicaciones, como el análisis de sentimientos en texto y la generación de texto natural. Por ejemplo, las MNN se pueden entrenar para reconocer el sentimiento en tweets y comentarios de redes sociales, lo que puede ser útil para identificar tendencias y opiniones públicas en tiempo real. También se utilizan en la generación de texto natural, como la creación de resúmenes y la generación de diálogos en chatbots.

Referencias: - Advanced Deep Learning with Python, Ivan Vasilev

7. ¿Qué son las Recurrent Neural Networks? Descripción. Propósito y Aplicaciones.

Las Recurrent Neural Networks (RNN) son un tipo de red neuronal que están diseñadas para procesar secuencias de datos, tales como series de tiempo, texto, audio, de forma rcurrente, es decir, “recordando” la información anterior. A diferencia de las redes neuronales convencionales, que procesan cada entrada de manera independiente. Las Recurrent Neural Networks (RNN) podrían ser comparadas con una persona que lee un libro y trata de entender la historia. En lugar de leer cada página de forma aislada, la persona intenta recordar la información clave de páginas anteriores para entender mejor la historia en su conjunto. Las RNN se utilizan en una variedad de aplicaciones, incluyendo el procesamiento del lenguaje natural, la generación de texto, la traducción automática, el reconocimiento de voz y la predicción de series de tiempo. Por ejemplo, una RNN entrenada en datos de texto se puede utilizar para generar texto natural, lo que puede ser útil en la creación de chatbots y sistemas de recomendación de texto, como chat GPT. Las RNN es que pueden tener dificultades para retener información de entradas muy antiguas debido a lo que se conoce como el problema del gradiente que explota o desaparece.

Referencias:

`Advanced Deep Learning with Python, Ivan Vasilev`

[35]:
```
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
!jupyter nbconvert --to pdf /content/drive/MyDrive/Colab_Notebooks/PEC2:
↪_Machine_Learning.ipynb
```

```
Reading package lists… Done
Building dependency tree
Reading state information… Done
texlive-fonts-recommended is already the newest version (2019.20200218-1).
texlive-plain-generic is already the newest version (2019.202000218-1).
texlive-xetex is already the newest version (2019.20200218-1).
0 upgraded, 0 newly installed, 0 to remove and 24 not upgraded.
[NbConvertApp] WARNING | pattern
'/content/drive/MyDrive/Colab_Notebooks/PEC2:_Machine_Learning.ipynb' matched no
files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
```

```
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--ClearOutputPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True
--TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True
--TemplateExporter.exclude_input=True
--TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the
system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful
for the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
```

```
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook',
'pdf', 'python', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
```

```
                can only be used when converting one notebook at a time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                to output to the directory of each notebook.
To recover
                                previous default behaviour (outputting to the
current
                                working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a
copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-
html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'rst', 'script', 'slides', 'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX
includes
```

```
            'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
            'classic'. You can specify the flavor of the format used.

            > jupyter nbconvert --to html --template lab mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```