Aprendizaje Profundo

Mario Agustín Sgró

Introducción

Se emplearán redes profundas y redes anidadas para resolver el problema de adopción de mascotas. Para ello se empleará una parte de la base de datos de la competencia PetFinder de Kaggle. Sucíntamente, lo que se busca es predecir la velocidad con la cual una mascota será adoptada teniendo en cuenta ciertas variables referidas a la misma como así también la descripción que se brinda de ella.

Base de Datos

One-Hot-Encoded variables

'Gender': género *(1 = Macho, 2 = Hembra, 3 = Mixed (grupos de mascotas))*
'Color1': color predominante
'MaturitySize': tamaño de adulto *(1 = Pequeño, 2 = Mediano, 3 = Grande, 4 = Extra grande, 0 = No especificado)*
'FurLength': largo del pelajo *(1 = Corto, 2 = Medio, 3 = Largo, 0 = No especificado)*
'Vaccinated': mascota vacunada? *(1 = Si, 2 = No, 3 = Ns/Nc)*
'Health': estado de salud *(1 = Saludable, 2 = Lesiones leves, 3 = Lesiones serias, 0 = No especificado)*
'Dewormed': mascota desparacitada? *(1 = Si, 2 = No, 3 = Ns/Nc)*
'Sterilized': mascota esterilizada? *(1 = Si, 2 = No, 3 = Ns/Nc)*

Embedded variables

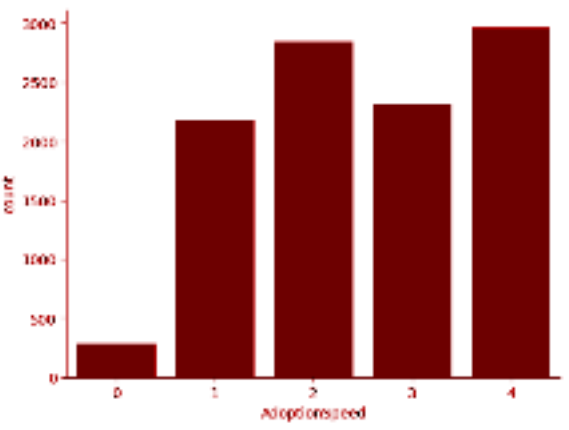
'Breed1': raza predominante
'Age': edad en meses
'Description': descripción

Target variable:

'AdoptionSpeed': descripción

Desbalance de clases

Se chequea el desbalance de clases de la variable objetivo. Se observa que la clase 0 tiene mucho desbalance respecto a las otras clases, esto será considerado al momento de ajustar el modelo definiendo pesos correspondientes a cada clase.



Variables

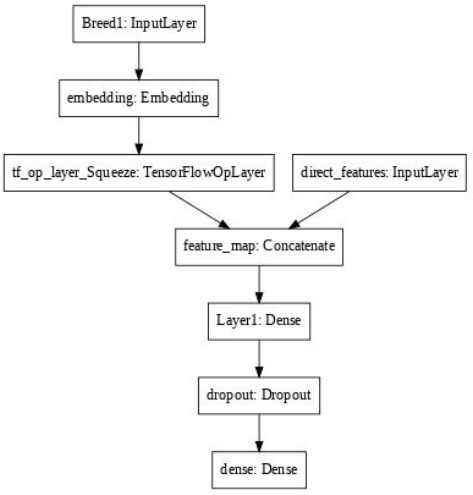
Las variables a utilizar serán:

```
# It's important to always use the same one-hot length
one_hot_columns = {
    one_hot_col: dataset[one_hot_col].max()
    for one_hot_col in ['Type','Gender', 'Color1', 'MaturitySize', 'FurLength',
                        'Vaccinated', 'Health', 'Dewormed', 'Sterilized']
}
embedded_columns = {
    embedded_col: dataset[embedded_col].max() + 1
    for embedded_col in ['Breed1']
}
numeric_columns = ['Fee','Age']
```

Donde a las variables englobadas en *one_hot_columns* se les aplica one-hot encoding por tratarse de variables categóricas. La variable *Breed1* también es categórica pero al tener un amplio rango de valores se prefiere aplicar un *embedding*. Las variables *Fee* y *Age* son numéricas y se les aplica una normalización [0,1].

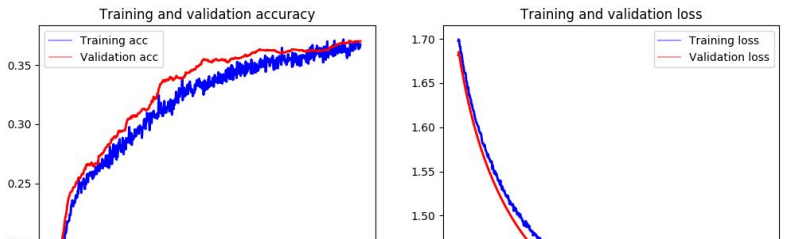
Modelo 1

Como primer modelo se considerará un MultiLayer Perceptron de una sola capa con N. El diagrama muestra un esquema de la red. Se considera además una capa de *Dropout* con el objetivo de evitar *overfitting*



Utilizando los siguientes parámetros se realiza un primer entrenamiento de la red:

```
params = {'batch_size': 4000,
          'hidden_layer_size_1': 64,
          'dropout_1': 0.2,
          'learning_rate': 0.0001,
          'optimizer': 'Adam'}
```

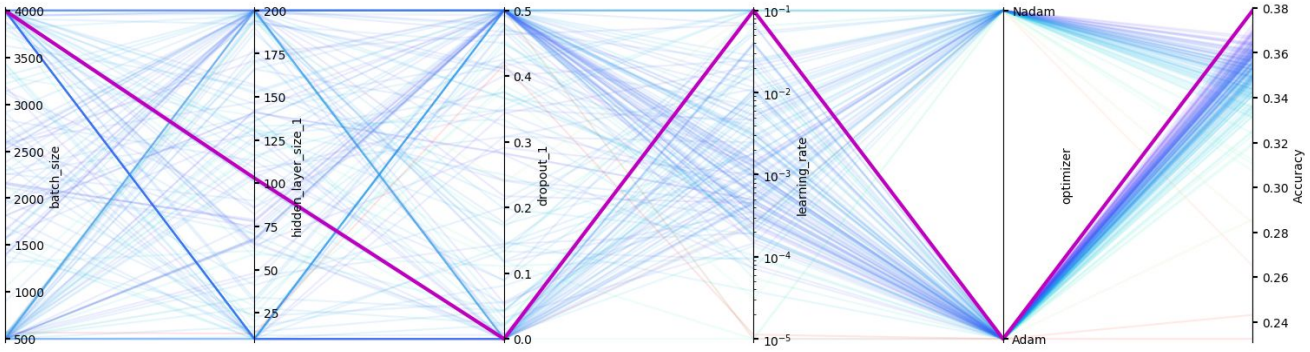


Estos paneles muestran la evolución de *accuracy* y de *loss* en función de la época de entrenamiento. El máximo valor de *accuracy* conseguido es de 0.35. Se podría entrenar algunas épocas más hasta observar que comienza a realizar *overfitting.*

Seguidamente se realiza un búsqueda de mejores parámetros empleando *skopt* con el espacio de parámetros definido como sigue:

```
search_space = {
  "batch_size": Integer(500, 4000, name="batch_size"),
  "hidden_layer_size_1": Integer(10, 200, name="hidden_layer_size"),
  "dropout_1": Real(low=0.0,high=0.5,prior='uniform',name='dropout'),
  "learning_rate": Real(low=1e-5, high=1e-1, prior='log-uniform', name='learning_rate'),
  "optimizer": Categorical(["Adam","Nadam"],name="optimizer")
}
```

Se realizaron 200 entrenamiento en dicho espacio, el gráfico a continuación muestra todos esos entrenamientos con los respectivos parámetros. Las líneas están coloreadas según el valor de *accuracy* obtenido para el correspondiente conjunto de parámetros. El mejor conjunto de parámetros es resaltado con la línea gruesa color magenta.

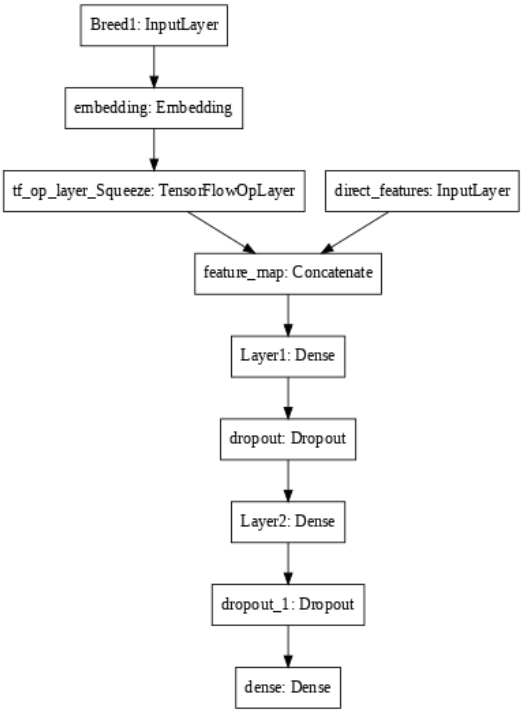


Mejor accuracy: 0.3789

Mejores parámetros: {'batch_size': 4000,
'hidden_layer_size_1': 103,
'dropout_1': 0.0,
'learning_rate': 0.1,
'optimizer': 'Adam'}

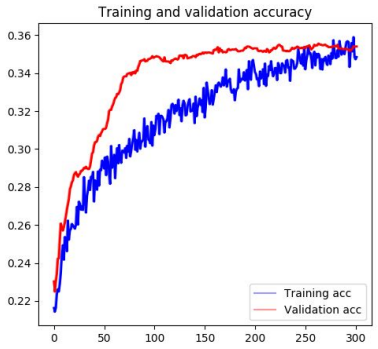
Modelo 2

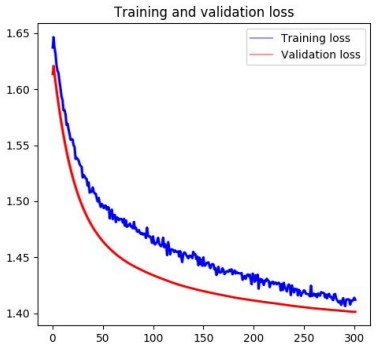
El segundo modelo que se considera es también un MultiLayer Perceptron pero esta vez con dos capas de N1 y N2 neuronas. El diagrama muestra un esquema de la red. Se considera además una capa de *Dropout* luego de cada capa de neuronas con el objetivo de evitar *overfitting.*



Utilizando los siguientes parámetros se realiza un primer entrenamiento de la red:

```
params = {'batch_size': 200,
          'hidden_layer_size_1': 128,
          'dropout_1': 0.2,
          'hidden_layer_size_2': 64,
          'dropout_2': 0.3,
          'learning_rate': 0.0001,
          'optimizer': 'Adam'}
```





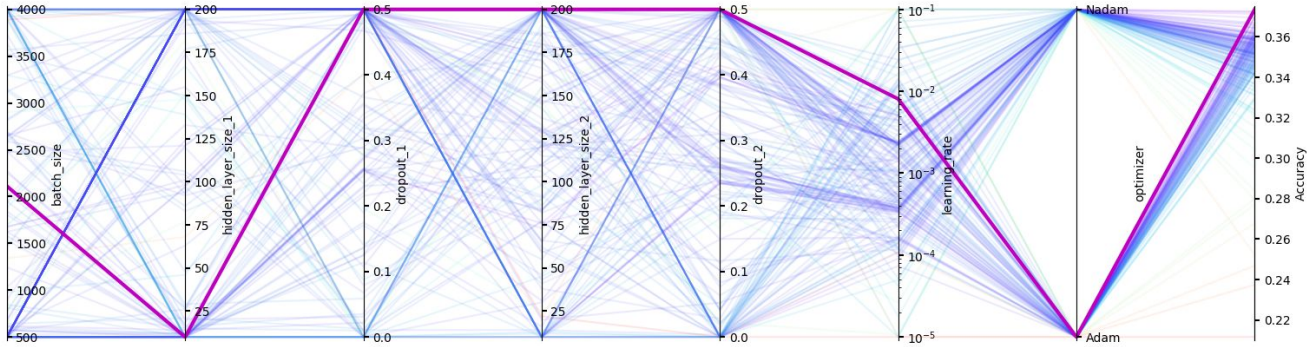
Estos paneles muestran la evolución de *accuracy* y de *loss* en función de la época de entrenamiento. El máximo valor de *accuracy* conseguido es de 0.35. Se podría entrenar algunas épocas más hasta observar que comienza a realizar *overfitting.*

Al igual que con el **Modelo1,** se realiza un búsqueda de mejores parámetros en el espacio de parámetros definido como sigue:

```
search_space = {
  "batch_size": Integer(500, 4000, name="batch_size"),
  "hidden_layer_size_1": Integer(10, 200, name="hidden_layer_size"),
  "dropout_1": Real(low=0.0,high=0.5,prior='uniform',name='dropout'),
  "hidden_layer_size_2": Integer(10, 200, name="hidden_layer_size"),
  "dropout_2": Real(low=0.0,high=0.5,prior='uniform',name='dropout'),
  "learning_rate": Real(low=1e-5, high=1e-1, prior='log-uniform', name='learning_rate'),
  "optimizer": Categorical(["Adam","Nadam"],name="optimizer")
}
```

Se realizaron 200 entrenamiento en dicho espacio, el gráfico a continuación muestra todos esos entrenamientos con los respectivos parámetros. Las líneas están coloreadas según el valor de *accuracy*

obtenido para el correspondiente conjunto de parámetros. El mejor conjunto de parámetros es resaltado con la línea gruesa color magenta.



Mejor accuracy: 0.3735

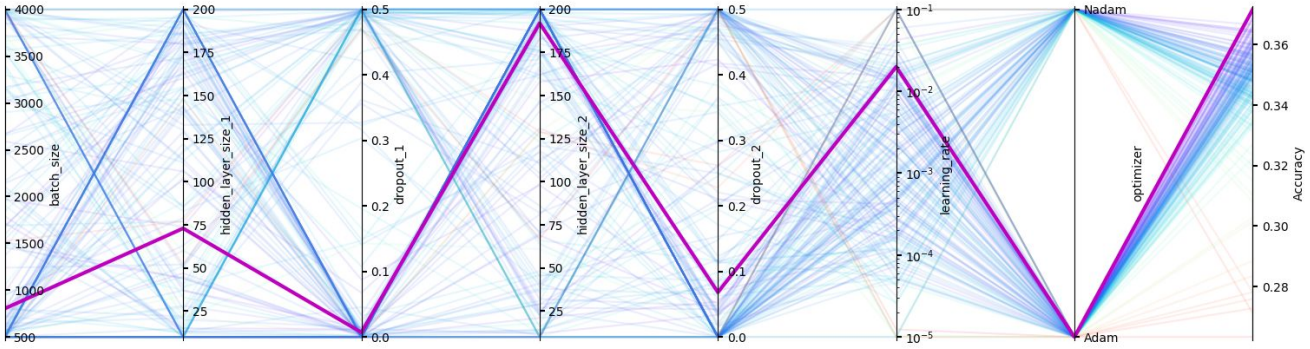
Mejores parámetros: {'batch_size': 2111,
'hidden_layer_size_1': 10,
'dropout_1': 0.5,
'hidden_layer_size_2': 200,
'dropout_2': 0.5,
'learning_rate': 0.008067009715798032,
'optimizer': 'Adam'}

Exploración sobre diferentes conjuntos de variables

Seguidamente se realizó un exploración considerando diferentes conjuntos de variables con el fin de observar cómo cambia la predicción del **Modelo2** con cada conjunto. Para ello se eligieron aleatoriamente dos variables del conjunto *one_hot_columns* mientras las otras variables se mantuvieron como estaban. Los resultados del entrenamiento sobre cada conjunto pueden observarse en la notebook. *A priori* el conjunto que parece brindar mejores resultados es:

```
# It's important to always use the same one-hot length
one_hot_columns = {
    one_hot_col: dataset[one_hot_col].max()
    for one_hot_col in ['Gender','Sterilized']
}
embedded_columns = {
    embedded_col: dataset[embedded_col].max() + 1
    for embedded_col in ['Breed1']
}
numeric_columns = ['Fee','Age']
```

La figura siguiente muestra la búsqueda de mejores parámetros para este conjunto de variables empleando el Modelo2 y su correspondiente espacio de parámetros. Las líneas están coloreadas según el valor de *accuracy* obtenido para el correspondiente conjunto de parámetros. El mejor conjunto de parámetros es resaltado con la línea gruesa color magenta.



Mejor accuracy: 0.3717

Mejores parámetros: {'batch_size': 804,
'hidden_layer_size_1': 73,
'dropout_1': 0.007025793769354307,
'hidden_layer_size_2': 192,
'dropout_2': 0.06819237944069574,
'learning_rate': 0.019888559306490722,
'optimizer': 'Adam'}

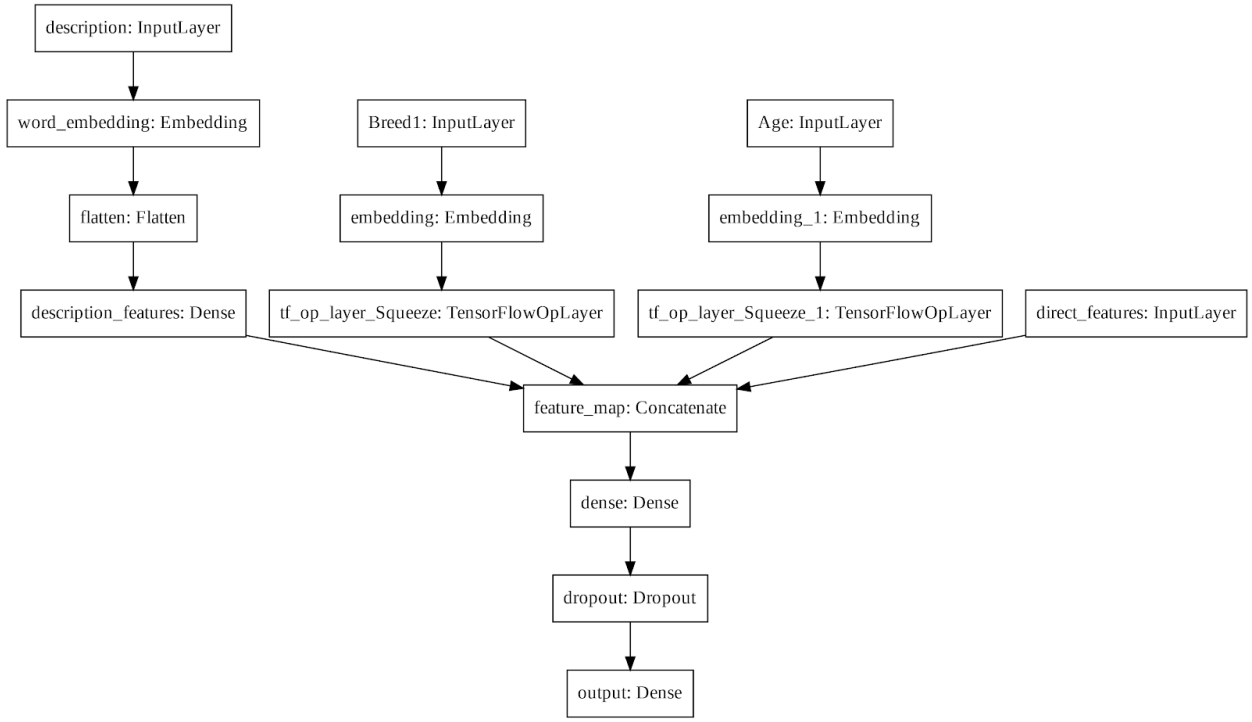
Conclusiones parciales

- Se encontraron similares empleando una red con una y dos capas de neuronas. Incluso puede decirse que la que tiene una sola capa “anda” mejor pues tiene menos parámetros para entrenar.
- Del análisis de los diferentes conjuntos de variables se puede concluir que es posible considerar menos variables y obtener los mismos resultados.
- Claramente, para ambos casos es necesaria una exploración más exhaustiva, ya se considerando otros diseños de modelos y otros conjuntos de variables (descartando/reemplazando las variables que se dejaron fijas por ejemplo).

Parte 2 : Redes anidadas

Modelo de base

El primer modelo considerado consiste en simplemente realizar un "*word embedding*" sobre el campo "*description*". Para se consideró el *embedding* de GloVe y tomando una longitud máxima del campo de **73** palabras ya que alrededor del **90%** de los datos tienen una longitud igual o menor a este valor.
El diagrama del modelo es mostrado en la siguiente figura:



Los **parámetros del modelo** son:
- Batch size
- Description features layer size
- Hidden layer size
- Dropout
- Learning rate
- Optimizer

Por ejemplo, considerando el siguiente conjunto de parámetros, se puede calcular la cantidad de pesos de la red a entrenar:

```
params = {'batch_size': 4000,
          'description_features_layer_size': 8,
          'hidden_layer_size': 128,
          'dropout': 0.5,
          'learning_rate': 0.0001,
          'optimizer': 'Adam'}
```

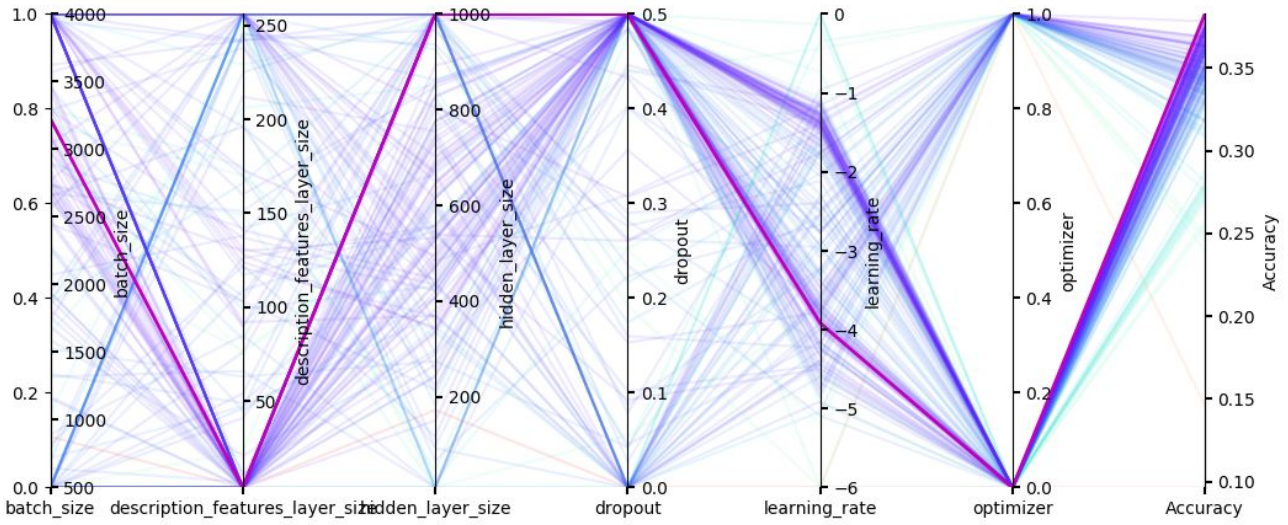
```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
description (InputLayer)        [(None, 73)]         0
__________________________________________________________________________________________________
Breed1 (InputLayer)             [(None, 1)]          0
__________________________________________________________________________________________________
Age (InputLayer)                [(None, 1)]          0
__________________________________________________________________________________________________
word_embedding (Embedding)      (None, 73, 100)      1000200     description[0][0]
__________________________________________________________________________________________________
embedding (Embedding)           (None, 1, 77)        23716       Breed1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 64)        16384       Age[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 7300)         0           word_embedding[0][0]
__________________________________________________________________________________________________
tf_op_layer_Squeeze (TensorFlow [(None, 77)]         0           embedding[0][0]
__________________________________________________________________________________________________
tf_op_layer_Squeeze_1 (TensorFl [(None, 64)]         0           embedding_1[0][0]
__________________________________________________________________________________________________
description_features (Dense)    (None, 8)            58408       flatten[0][0]
__________________________________________________________________________________________________
direct_features (InputLayer)    [(None, 29)]         0
__________________________________________________________________________________________________
feature_map (Concatenate)       (None, 178)          0           tf_op_layer_Squeeze[0][0]
                                                                 tf_op_layer_Squeeze_1[0][0]
                                                                 description_features[0][0]
                                                                 direct_features[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 128)          22912       feature_map[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 128)          0           dense[0][0]
__________________________________________________________________________________________________
output (Dense)                  (None, 5)            645         dropout[0][0]
==================================================================================================
Total params: 1,122,265
Trainable params: 122,065
Non-trainable params: 1,000,200
```

Búsqueda de mejores parámetros

Definido el modelo se realizó un búsqueda de los mejores parámetros. Para ello se empleó la rutina *skopt.* El espacio de parámetros se definió según el siguiente diccionario:

```
search_space = {
    "batch_size": Integer(500,4000),
    "description_features_layer_size": Integer(4, 256),
    "hidden_layer_size": Integer(10, 1000),
    "dropout": Real(low=0.0, high=0.5, prior='uniform'),
    "learning_rate": Real(low=1e-6, high=1.0, prior='log-uniform'),
    "optimizer": Categorical(["Adam","Nadam"])
    }
```

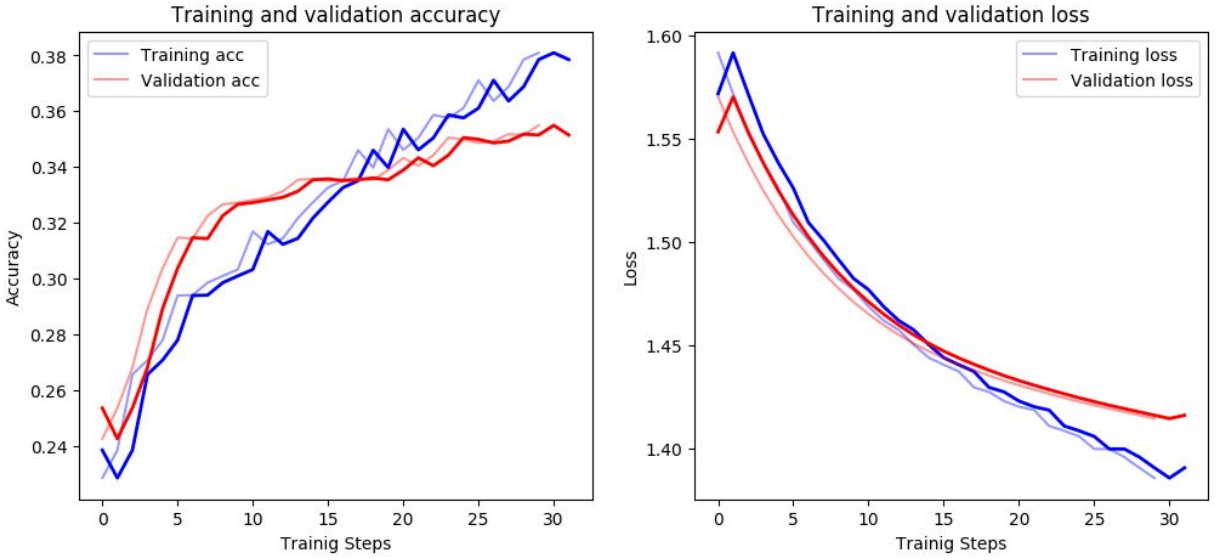
El siguiente gráfico es una representación de los diferentes modelos ejecutados durante este proceso. Cada línea representa el conjunto de datos que describe sobre los ejes verticales, el valor del *accuracy* obtenido es graficado en el último eje a la derecha. Cada línea está coloreada según el valor de *accuracy* obtenido. El conjunto de mejores valores está representado por la línea gruesa magenta.



Mejor accuracy: 0.38270

Mejores parámetros: {'batch_size': 3225,
'description_features_layer_size': 4,
'hidden_layer_size': 1000,
'dropout': 0.5,
'learning_rate': 1.216E-4,
'optimizer': 'Adam'

En el siguiente gráfico se muestra la evolución del *accuracy* y la función de *loss* durante el entrenamiento de 30 épocas del modelo construido utilizando los mejores valores para los parámetros.

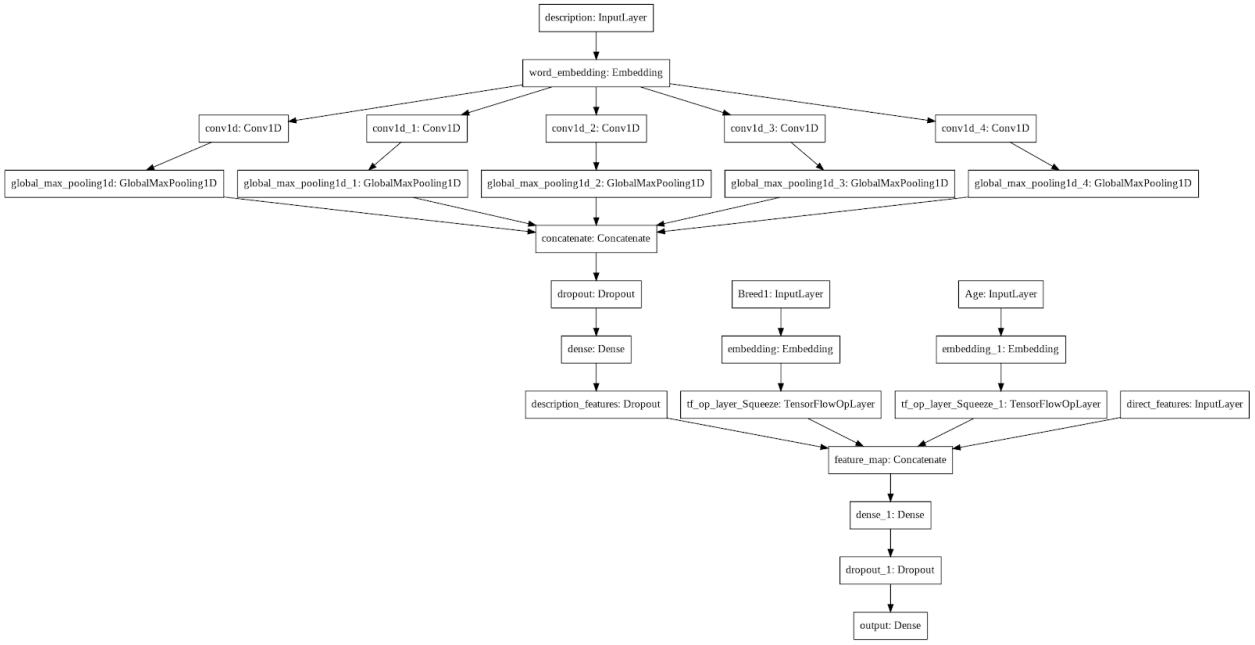


Puede observarse que a partir de transcurridas 16 épocas el modelo comienza a sobre-ajustar los datos de entrenamiento. Esto puede observarse porque el *accuracy* sobre los datos de entrenamiento sigue "mejorando" más rápidamente que sobre los datos de validación.

Modelo de Redes Escalonadas

1.- Redes convolucionales

Partiendo del modelo anterior se le agregó a la rama con los *embeddings* del campo descripción una red convolucional. Ésta última consistente de **5** filtros convolucionales con un tamaño de **200xN**, donde **N=2,3,4,5 o 6**, según el filtro. Luego de esta capa convolucional se coloca una capa densa y una capa de *dropout* antes de concatenar nos los resto de las propiedades.
El diagrama del modelo es mostrado en la siguiente figura:



Los **parámetros del modelo** son:

- Bach size
- Description features layer size
- Dropout 1
- Dropout 2
- Hidden layer size
- Dropout
- Learning rate
- Optimizer

Por ejemplo, considerando el siguiente conjunto de parámetros, se puede calcular la cantidad de pesos de la red a entrenar:

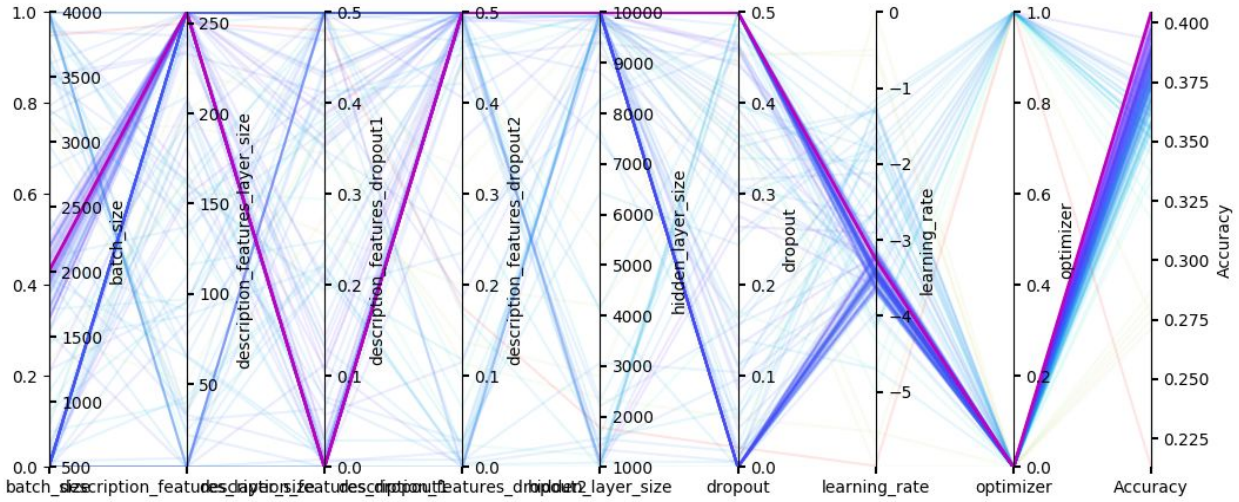
```
params = {'batch_size': 4000,
          'description_features_layer_size': 512,
          'description_features_dropout1': 0.2,
          'description_features_dropout2': 0.2,
          'hidden_layer_size': 1000,
          'dropout': 0.5,
          'learning_rate': 0.001,
          'optimizer': 'Adam'}
```

Layer (type)	Output Shape	Param #	Connected to
description (InputLayer)	[(None, 73)]	0	
word_embedding (Embedding)	(None, 73, 100)	1000200	description[0][0]
conv1d (Conv1D)	(None, 72, 200)	40200	word_embedding[0][0]
conv1d_1 (Conv1D)	(None, 71, 200)	60200	word_embedding[0][0]
conv1d_2 (Conv1D)	(None, 70, 200)	80200	word_embedding[0][0]
conv1d_3 (Conv1D)	(None, 69, 200)	100200	word_embedding[0][0]
conv1d_4 (Conv1D)	(None, 68, 200)	120200	word_embedding[0][0]
global_max_pooling1d (GlobalMax	(None, 200)	0	conv1d[0][0]
global_max_pooling1d_1 (GlobalM	(None, 200)	0	conv1d_1[0][0]
global_max_pooling1d_2 (GlobalM	(None, 200)	0	conv1d_2[0][0]
global_max_pooling1d_3 (GlobalM	(None, 200)	0	conv1d_3[0][0]
global_max_pooling1d_4 (GlobalM	(None, 200)	0	conv1d_4[0][0]
concatenate (Concatenate)	(None, 1000)	0	global_max_pooling1d[0][0] global_max_pooling1d_1[0][0] global_max_pooling1d_2[0][0] global_max_pooling1d_3[0][0] global_max_pooling1d_4[0][0]
Breed1 (InputLayer)	[(None, 1)]	0	
Age (InputLayer)	[(None, 1)]	0	
dropout (Dropout)	(None, 1000)	0	concatenate[0][0]
embedding (Embedding)	(None, 1, 77)	23716	Breed1[0][0]
embedding_1 (Embedding)	(None, 1, 64)	16384	Age[0][0]
dense (Dense)	(None, 512)	512512	dropout[0][0]
tf_op_layer_Squeeze (TensorFlow	[(None, 77)]	0	embedding[0][0]
tf_op_layer_Squeeze_1 (TensorFl	[(None, 64)]	0	embedding_1[0][0]
description_features (Dropout)	(None, 512)	0	dense[0][0]
direct_features (InputLayer)	[(None, 29)]	0	
feature_map (Concatenate)	(None, 682)	0	tf_op_layer_Squeeze[0][0] tf_op_layer_Squeeze_1[0][0] description_features[0][0] direct_features[0][0]
dense_1 (Dense)	(None, 1000)	683000	feature_map[0][0]
dropout_1 (Dropout)	(None, 1000)	0	dense_1[0][0]

```
output (Dense)                  (None, 5)            5005        dropout_1[0][0]
==================================================================================================
Total params: 2,641,817
Trainable params: 1,641,617
Non-trainable params: 1,000,200
__________________________________________________________________________________________________
```

Al igual que antes, la exploración para encontrar los mejores parámetros se realizó sobre el siguiente espacio de parámetros y el resultado de dicho proceso está sintetizado en la figura.

```
search_space = {
        "batch_size": Integer(500,4000, name="batch_size"),
        "description_features_layer_size": Integer(4, 256,
                                                   name="description_features_layer_size"),
        "description_features_dropout1": Real(low=0.0, high=0.5, prior='uniform',
                                              name='description_features_dropout1'),
        "description_features_dropout2": Real(low=0.0, high=0.5, prior='uniform',
                                              name='description_features_dropout2'),
        "hidden_layer_size": Integer(1000, 10000, name="hidden_layer_size"),
        "dropout": Real(low=0.0, high=0.5, prior='uniform', name='dropout'),
        "learning_rate": Real(low=1e-6, high=1.0, prior='log-uniform',
                              name='learning_rate'),
        "optimizer": Categorical(["Adam","Nadam"], name="optimizer")
}
```

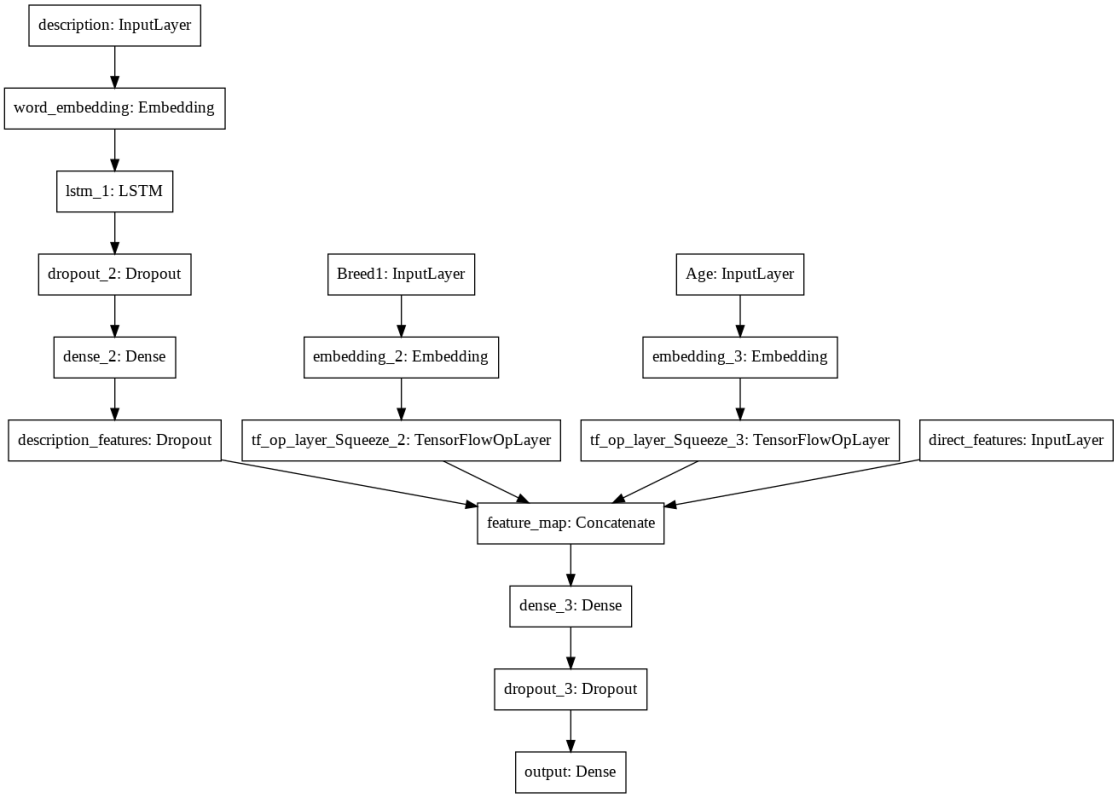


Mejor accuracy: 0.4044

Mejores parámetros: {'batch_size': 2006,
'description_features_layer_size': 256,
'description_features_dropout1': 0.0,
'description_features_dropout2': 0.5,
'hidden_layer_size': 10000,
'dropout': 0.5,
'learning_rate': 0.0005562695423941707,
'optimizer': 'Adam'}

2.- Long Short-Term Memory (LSTM)

Como otra alternativa se emplea el esquema LSTM para modelar la sección de la red "encargada" de los *embeddings* del campo descripción. LSTM fue diseñada para procesar secuencias de datos, al mismo tiempo que evita los problemas de los gradientes que se hacen o muy grandes o muy chicos. Luego de esta red se coloca una capa de dropout, una capa densa y una capa de *dropout* antes de concatenar nos los resto de las propiedades.
El diagrama del modelo es mostrado en la siguiente figura:



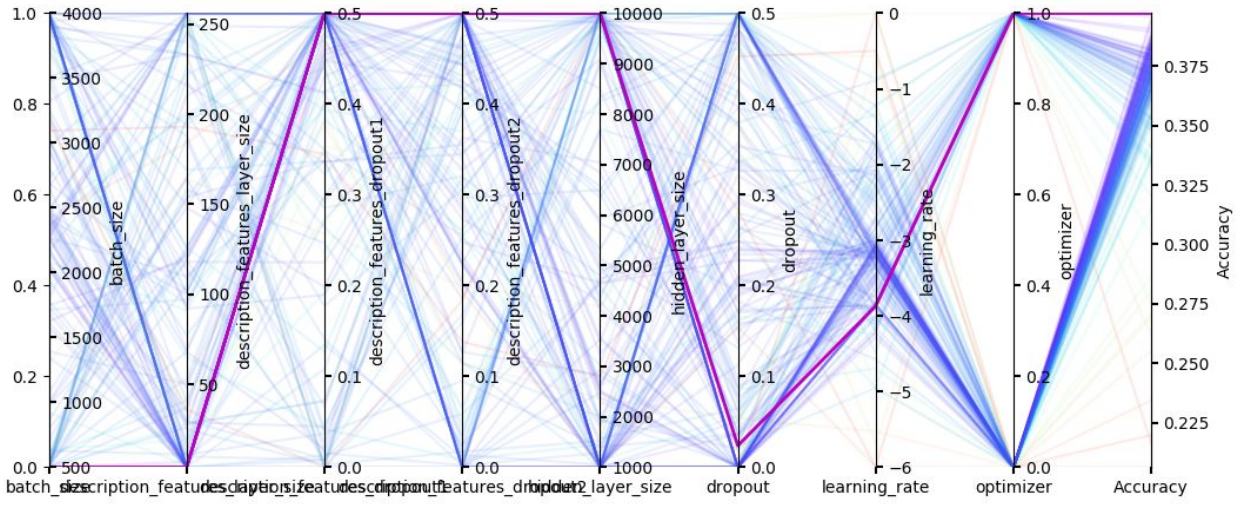

```
Model: "LSTM"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
description (InputLayer)        [(None, 73)]         0
__________________________________________________________________________________________________
word_embedding (Embedding)      (None, 73, 100)      1000200     description[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 128)          117248      word_embedding[0][0]
__________________________________________________________________________________________________
Breed1 (InputLayer)             [(None, 1)]          0
__________________________________________________________________________________________________
Age (InputLayer)                [(None, 1)]          0
__________________________________________________________________________________________________
dropout (Dropout)               (None, 128)          0           lstm[0][0]
__________________________________________________________________________________________________
embedding (Embedding)           (None, 1, 77)        23716       Breed1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 64)        16384       Age[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 256)          33024       dropout[0][0]
__________________________________________________________________________________________________
tf_op_layer_Squeeze (TensorFlow [(None, 77)]         0           embedding[0][0]
```

```
tf_op_layer_Squeeze_1 (TensorFl [(None, 64)]         0           embedding_1[0][0]
description_features (Dropout)  (None, 256)          0           dense[0][0]
direct_features (InputLayer)    [(None, 29)]         0
feature_map (Concatenate)       (None, 426)          0           tf_op_layer_Squeeze[0][0]
                                                                 tf_op_layer_Squeeze_1[0][0]
                                                                 description_features[0][0]
                                                                 direct_features[0][0]
dense_1 (Dense)                 (None, 1000)         427000      feature_map[0][0]
dropout_1 (Dropout)             (None, 1000)         0           dense_1[0][0]
output (Dense)                  (None, 5)            5005        dropout_1[0][0]
==================================================================================================
Total params: 1,622,577
Trainable params: 622,377
Non-trainable params: 1,000,200
```

El espacio sobre el que se buscan los mejores parámetros está definido como sigue:

```
search_space = {
    "batch_size": Integer(500,4000, name="batch_size"),
    "description_features_layer_size": Integer(4, 256,
                                               name="description_features_layer_size"),
    "description_features_dropout1": Real(low=0.0, high=0.5, prior='uniform',
                                          name='description_features_dropout1'),
    "description_features_dropout2": Real(low=0.0, high=0.5, prior='uniform',
                                          name='description_features_dropout2'),
    "hidden_layer_size": Integer(1000, 10000, name="hidden_layer_size"),
    "dropout": Real(low=0.0, high=0.5, prior='uniform', name='dropout'),
    "learning_rate": Real(low=1e-6, high=1.0, prior='log-uniform',name='learning_rate'),
    "optimizer": Categorical(["Adam","Nadam"], name="optimizer")
}
```

Resumen del proceso de búsqueda de mejores parámetros:



Mejor accuracy: 0.3972

Mejores parámetros: {'batch_size': 500,
'description_features_layer_size': 4,
'description_features_dropout1': 0.5,
'description_features_dropout2': 0.5,
'hidden_layer_size': 10000,
'dropout': 0.02376663393246057,
'learning_rate': 0.00013451014355483958,
'optimizer': 'Nadam'}

Conclusiones parciales

- Se encuentra que utilizando el campo de descripción y empleando un método de *word embedding* se obtienen mejores resultados comparando con los obtenidos en el práctico 1.
- Así mismo, el empleo de redes anidadas (convolucionales o LSTM, en este caso) muestra mejores resultados. Aunque las opciones aquí empleadas dejan la sensación que se ha intentado atacar una hormiga con un bazooka :).
- Apesar de todo lo empleado no se ha podido conseguir una *accuracy* mayor al 0.41.