Sumario

[11](#__RefHeading___Toc603_4015247920)

[Implicaciones del batch\_size 12](#__RefHeading___Toc605_4015247920)

[Implicaciones de la dimensión de embeddings 13](#__RefHeading___Toc607_4015247920)

[13](#__RefHeading___Toc609_4015247920)

[Base 1 capa 14](#__RefHeading___Toc611_4015247920)

[Efecto en el número de celdas, batch y reg en una capa 15](#__RefHeading___Toc613_4015247920)

[Efecto del gradient clipping 16](#__RefHeading___Toc615_4015247920)

[Efecto del número de capas 17](#__RefHeading___Toc617_4015247920)

[Efecto de la regularización en dos capas 17](#__RefHeading___Toc619_4015247920)

[Redes Conv1D 21](#__RefHeading___Toc621_4015247920)

[LSTM and Convolution Networks exploration for Parkinson’s Diagnosis 22](#__RefHeading___Toc623_4015247920)

[Lstm, conv y conv → stack. El que mejor funciona CONV1D 22](#__RefHeading___Toc625_4015247920)

[22](#__RefHeading___Toc627_4015247920)

Data

2.5M series de utilizaciones de suelo durante nueve años.

Modelo base

Embeddings utilizando secuencia completa.

train/test split: 30%

Optimización de la secuencia

\* Reduce number of parameters of the embedding; embedding\_size = vocab\_size/2

Loss on test data: [2.915931224822998, 0.4563530683517456]

La red está sobre ajustando, modificar parámetros

\* La proporción sigue muy imbalanced, increase the rate for the underrepresented categories

\* Probar con diferenes tamaños de secuencia

\* Chage split to structured and increase test split to 0.3

\* tune the batch size

\* Change activation to relu

Hay un problma con el relu, si se utiliza una función de activación unbounded, los pesos dentro de las células LSTM se escapan a infinito, la función de pérdida se dispara y el modelo pierde precisión. Opciones:

- Normalización batch antes de la célula LSTM si antes viene una salida outbounded

- Normalización batch antes de la activación, desactivar de la céluna y poner esquema:

LSTM > batch > activación (relu)

===Train/test size: 14432, 6186

Número de características: 6

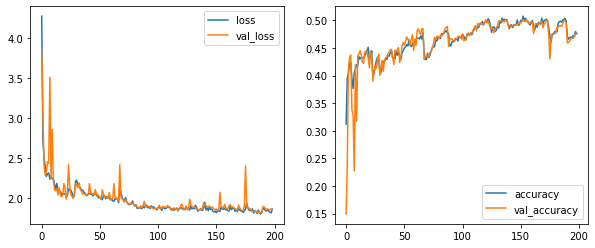
Clasificador base:

|  |
| --- |
| **Modelo base**  Lanzamos TPOT para obtener un clasificador automático  Best pipeline: ExtraTreesClassifier(input\_matrix, bootstrap=False, criterion=entropy, max\_features=0.7500000000000001, min\_samples\_leaf=10, min\_samples\_split=13, n\_estimators=100)  0.5094219775566378 |
| **Reducir número de celdas de LSTM**, reducir el número de celdas para evitar que el modelo sobreajuste a los datos, bajar de 50 a 30  model.add(Embedding(vocab\_size, embedding\_size,input\_length=sequence\_length)) model.add(LSTM(**30**)) model.add(Dense(vocab\_size, activation='sigmoid')  Epoch 220/300  loss: 1.0106  accuracy: 0.6650  val\_loss: 1.7342  val\_accuracy: 0.5099  Conclusiones  La red se siguen sobreajustando, aumenta el tiempo de entrenamiento. |
| **Añadir dropout para reducir overfitting**  Modelo:  model.add(Embedding(vocab\_size, embedding\_size, input\_length=sequence\_length)) model.add(LSTM(30, **dropout=0.3**)) model.add(Dense(vocab\_size, activation='sigmoid'))  Epoch 251/300 loss: 1.2098  accuracy: 0.5962 -  val\_loss: 1.5169  val\_accuracy: 0.5233  Conclusiones  La red se siguen sobreajustando, aumenta el tiempo de entrenamiento. |
| **Reducir tamaño de vocabulario a la mitad**  Reducir las dimensiones de representación de las secuencias de códigos para que tengan menos dimensiones. Bajar el embedding\_size a lamitad del vocabulario.  Epoch 200/200  loss: 1.0741  accuracy: 0.6452  val\_loss: 1.6498  val\_accuracy: 0.5118  Conclusiones  No tiene un efecto significativo claro, pero parece que provoca más sobreajuste, el % de acierto sobre el conjunto de entrenamiento sube y baja sobre el de test. |
| **Añadir capa de normalización antes de la capa densa**  model.add(LSTM(30)) model.add(BatchNormalization()) model.add(Dense(vocab\_size, activation='sigmoid')) print(model.summary())  Epoch 150/150  loss: 1.1742 -  accuracy: 0.5996 -  val\_loss: 1.6011 -  val\_accuracy: 0.5338  Conclusiones  Mejora muy escasa en la función de pérdida |
| **Cambiar función de activación a RELU en capa densa**  Se produce el efecto contrario, las métricas de train/test van paralelas y la función de pérdida decrece muy lentamente. El modelo mejora significativamente.  model.add(Embedding(vocab\_size, embedding\_size, input\_length=sequence\_length)) model.add(LSTM(30, dropout=0.3)) model.add(BatchNormalization()) model.add(Dense(vocab\_size, activation='relu'))  Results file: training\_history\_6\_1594539140.00881.csv  Last 10 records avg:  epoch 263.500000  accuracy 0.483107  loss 1.782723  val\_accuracy 0.480925  val\_loss 1.790821  Hay que incrementar la capacidad del modelo y bajamos el dropout a 0.2 y aumentamos a 200 epocas |
| **Aumentar capacidad del modelo y número de épocas**  Aumentamos el número de unidades de la capa LSTM y el dropout  model.add(Embedding(vocab\_size, embedding\_size, input\_length=sequence\_length)) model.add(LSTM(**50, dropout=0.3**)) model.add(BatchNormalization()) model.add(Dense(vocab\_size, activation='relu'))  Last 10 records avg:  epoch 294.500000  accuracy 0.499515  loss 1.726744  val\_accuracy 0.484853  val\_loss 1.807326  No hay mejora significativa, volvemos a 30 unidades    Aumetnar o reducir el drop out tiene efectos negativos, reduciendo a 0.2 o aumentando a 0.4, la respuesta de las métricas es errática.  Con un dropout de 40%: |
| **Cambiar tamaño del batch size para ver cómo influye en el entrenamiento**  Al cambiar el tamaño del batch\_size está mejorando el entrenamiento, esto se tiene que deber a que las clases están muy desbalanceadas, probar con un dataset nuevo que equilibre al menos el número de clases de y.  Fijamos 128 y creamos nuevo dataset  model.add(LSTM(30, dropout=0.3)) model.add(BatchNormalization()) model.add(Dense(vocab\_size, activation='relu')) |
| **Cambiar función de activación en LSTM a RELU**  model.add(Embedding(vocab\_size, embedding\_size, input\_length=sequence\_length)) model.add(LSTM(30, dropout=0.3, activation=None)) model.add(BatchNormalization()) model.add(Activation("relu")) model.add(Dense(vocab\_size, activation='relu'))  Mejora pero se sigue quedando estancado en torno al 55% hay que mejorar la capacidad del modelos.    Last 10 records avg:  epoch 224.500000  accuracy 0.529968  loss 1.761819  val\_accuracy 0.525994  val\_loss 1.799849 |
| **Crear conjunto de datos balanceado**  Utilizando undersampler para clases mayoritarias para crear un conjunto balanceado respecto a la categoría.  epoch 300  accuracy 0.408273  loss 2.170180  val\_accuracy 0.414272  val\_loss 2.154978  epoch 300  accuracy 0.443459  loss 1.974530  val\_accuracy 0.453401  val\_loss 1.925942   |  |  | | --- | --- | | Distrución de clases en dataset original | Distribución de clases en dataset balanceado |   Tiene efecto negativo, se ha reducido el dataset a la mitad.    Probamos de nuevo aumentando el número de mínimo de muestras a 1000   |  |  | | --- | --- | |  |  | |
| **Aumentar el número de celdas internas**  Aumentar número de unidades a 40  **epoch 300**  accuracy 0.470519  loss 1.855217  val\_accuracy 0.478633  val\_loss 1.877754  dtype: float64    Con 50 unidades empieza a aparecer el sobreajuste  epoch 300  accuracy 0.493722  loss 1.787970  val\_accuracy 0.498817  val\_loss 1.951795  dtype: float64 |
| **Cambiar función de ajuste de categorical cross-entropy to f1-score**  El aprendizaje es más estable, pero se llega a una tasa en la que la red empieza a sobrentrenar. Antes de empezar con las optimizaciones, hay que intentar mejorar la capacidad del modelo.  epoch 100  f1 0.726099  loss 0.273565  val\_f1 0.686879  val\_loss 0.313024 |
| **Parametrización LSTM**   * Recurrent drop-out: 0.2 y mover dropout a capa inpendendiente antes del dense. → No influye   model = Sequential() model.add(Embedding(vocab\_size, embedding\_size,input\_length=sequence\_length)) model.add(LSTM(40, **recurrent\_dropout=0.2**)) model.add(Dropout(0.2)) model.add(BatchNormalization()) model.add(Dense(vocab\_size, activation='relu'))  epoch 100  f1 0.726099  loss 0.273565  val\_f1 0.686879  val\_loss 0.313024    \* Cambiar dropout antes de la normalización batch: sin efecto  epoch 26.500000  f1 0.694709  loss 0.304934  val\_f1 0.671864  val\_loss 0.327877  dtype: float64 |
| **Utilizar modelo con stacked LSTM**  model.add(Embedding(vocab\_size, embedding\_size, input\_length=sequence\_length)) model.add(LSTM(32, return\_sequences=True)) model.add(BatchNormalization()) model.add(LSTM(64)) # , recurrent\_dropout=0.2 model.add(BatchNormalization()) model.add(Dense(vocab\_size, activation='relu'))  Aumenta el score sobre el conjunto de entrenamiento pero no sobre el conjunto de test, el modelo se sobreajusta a los datos.   |  |  | | --- | --- | | LSTM one layer | LSTM two layers | |
| **Utilizar capa TimeDistributed**  No tiene sentido en este caso, cada elemento temporal de la secuencia es una de las dimensiones del embedding, no tiene sentido aplicar una red interca sobre cada columna por separado. |
| **Modificar embeddings**  Cambiar la dimensión de la capa de embeddings no tiene efecto, parace que son pocas combinaciones y pocas columnas para este tipo de técnica.  Haciendo un plot de los pesos de la capa de embeddings y proyectándolo en dos dimensiones con TSNE parecen equidistantes, por lo que lo se esan encontrando similitudes en**tre los cultivos, el lstm está recibiendo básicamente la misma entrada.**  Conclusiones: utilizamos dimensión 5. |
| **Reducción programada del learning rate**  \* Como inversa de exponencial: sin efecto  \* Drop en base a épocas mal resultado |
| **Uso de redes de convolución 1D**  Uso de filtros 1D para extraer patrones en las secuencias temporales.  2 redes de convolución con dropout y global average  20200718\_094307\_con1d\_1l\_emb5   |  |  | | --- | --- | |  | epoch 87.500000  f1 0.606053  loss 0.393834  val\_f1 0.607326  val\_loss 0.391150 | |
| **Red de convolución 1D con LSTM por encima**  20200718\_161056\_lstm\_2l\_64\_32\_16\_emb5   |  |  | | --- | --- | |  | epoch 87.500000  f1 0.672397  loss 0.327495  val\_f1 0.665538  val\_loss 0.332781  dtype: float64 | |
| **Red LSTM de tres capas**  689/689 - 49s - loss: 0.3248 - f1: 0.6751 - val\_loss: 0.4267 - val\_f1: 0.5719  Epoch 00038: early stopping  Loss on test data: [0.4267488121986389, 0.5719085335731506]  **20200718\_161056\_lstm\_2l\_64\_32\_16\_emb5**   |  |  | | --- | --- | |  | **epoch 92.500000**  f1 0.664051  loss 0.335830  val\_f1 0.657944  val\_loss 0.340288  dtype: float64 | |
| **Red 1D + LSTM utilizando redes de restos**   |  |  | | --- | --- | |  | epoch 39.500000  f1 0.705121  loss 0.294734  val\_f1 0.678709  val\_loss 0.319534 |  |  |  | | --- | --- | |  | epoch 49.500000  f1 0.715732  loss 0.284151  val\_f1 0.684259  val\_loss 0.313946  Aquí hay que añadir dropOut o normBatch, la red se está sobresajustando |   **Conclusión**  Las redes con 1D consiguen llegar a valores máximos mucho antes que solo LSTM. Los resultados son similares, no consiguen mejorar una red LSTM de dos niveles   |  |  | | --- | --- | |  | 0.7649 / 0.73432 | |  | 0.7866 - val\_loss: 0.2624 - val\_f1: 0.7372 | |  | 0.76, 0.730 | |  | 0.76, 0.735 | |  | La conv1D está traspuesta, está aprendiendo patrones sobre el R20 de los embeddings en lugar de la secuencia.  Con 64 unids en LSTM  loss: 0.1902 - f1: 0.8094 - val\_loss: 0.2732 - val\_f1: 0.7264  Con 12 unid.  loss: 0.1883 - f1: 0.8113 -  val\_loss: 0.2679 - val\_f1: 0.7318  Con 128 unid.  loss: 0.1806 - f1: 0.8213 - val\_loss: 0.2695 - val\_f1: 0.7329  \* Quitando rama lstm  loss: 0.1823 - f1: 0.8174 - val\_loss: 0.2644 - val\_f1: 0.7356  \* Añadiendo conv1d como feature extraction para la lstm  loss: 0.2180 - f1: 0.7842 - val\_loss: 0.2748 - val\_f1: 0.7278  \* Flatten de embeddings  loss: 0.2333 - f1: 0.7663 - val\_loss: 0.2660 - val\_f1: 0.7339 | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |   Aumentando los filtros a 128 no mejora. |
| LSTM-FCM  <https://towardsdatascience.com/lstm-fcn-for-cardiology-22af6bbfc27b> |
| **Parametrización del optimizador**  **Gradient Clipping**  (lr=0.001, clipvalue=0.3),  No hay mejora  **Learning rate:** |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

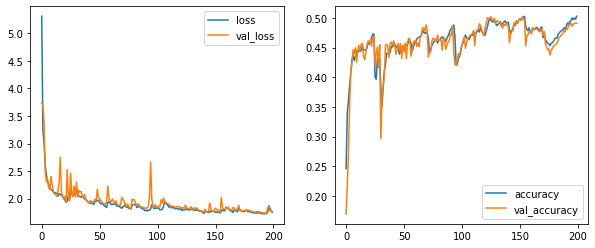
# 

# Implicaciones del batch\_size

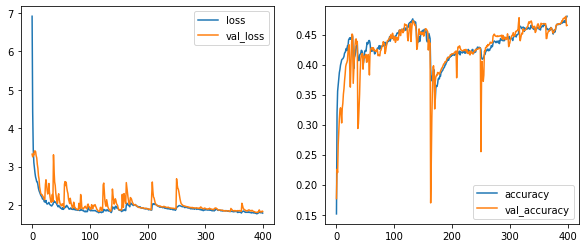
64



128



512



# Implicaciones de la dimensión de embeddings

La profundidad del embedding mejora el entrenamiento.

Tomando una red de 128 unidades, entrenamos para profundidades de 5,12,20

detph 5 (0.7/0.71) > detph 12 (0.73/0.72) > **detph 20 (0.74/0.73)** > depth 25 (0.75, 0.72)

# 

\* Probar con profundidad 25

|  |  |
| --- | --- |
| Embeddings size 5 – 40 epocas | Embeddings size 10 -40 epocas |
|  |  |
| 100 epocas | 100 epocas |
|  |  |
|  |  |

# Base 1 capa

256 embeddings de 20 con batch y sin regularización. 0.78,0.73.

dropout=0.3, recurrent\_dropout=0.4 clipping: 0.3, lr:0.001

Sin callback para lr dinámica

**trail f1: 0.8004 - val\_loss: 0.2635 - val\_f1: 0.7363**

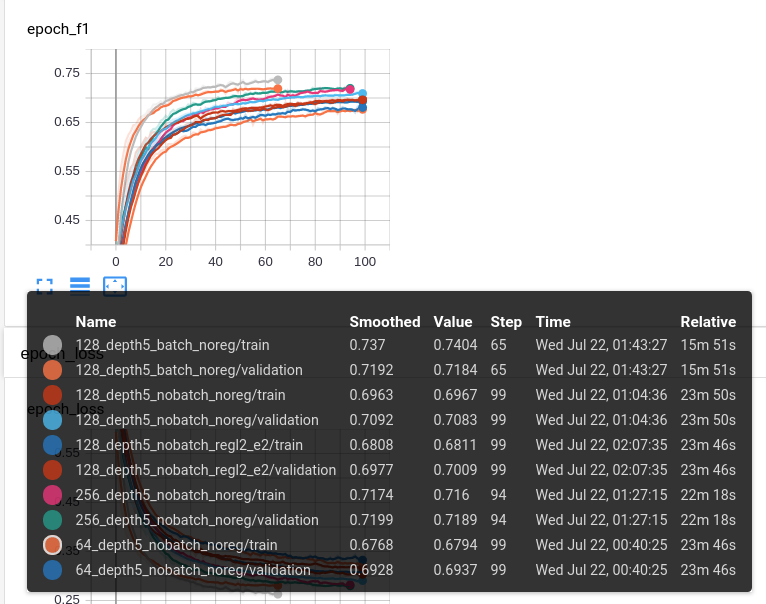
Optimización bayesiana:

{'target': 0.7342096567153931, 'params': {'dropout': 0.37460037107263344, 'opt\_clipvalue': 0.44211898516408055, 'opt\_lr': 0.0009835405503902273, 'recurrent\_dropout': 0.40005772597798706}}

# Efecto en el número de celdas, batch y reg en una capa

1. Con una única capa, añadimos celdas para ver cuál funciona mejor.
2. Una capa de 128 unidades que tomamos como base y sin regularización.
3. Conclusiones:

* El aumento en el número de neuronas mejora 256 (0.72/0.72) > 128 (0.7/0.71) > 64 (0.68/0.68)
* La capa batch reduce el tiempo de entrenamiento 128 batch (0.74/0.72) > 128 no batch (0.7/0.71)
* La regularización recurrente tiene un efecto negativo: 128 no reg (0.7/0.71) > 128 reg (0.68, 0.69)

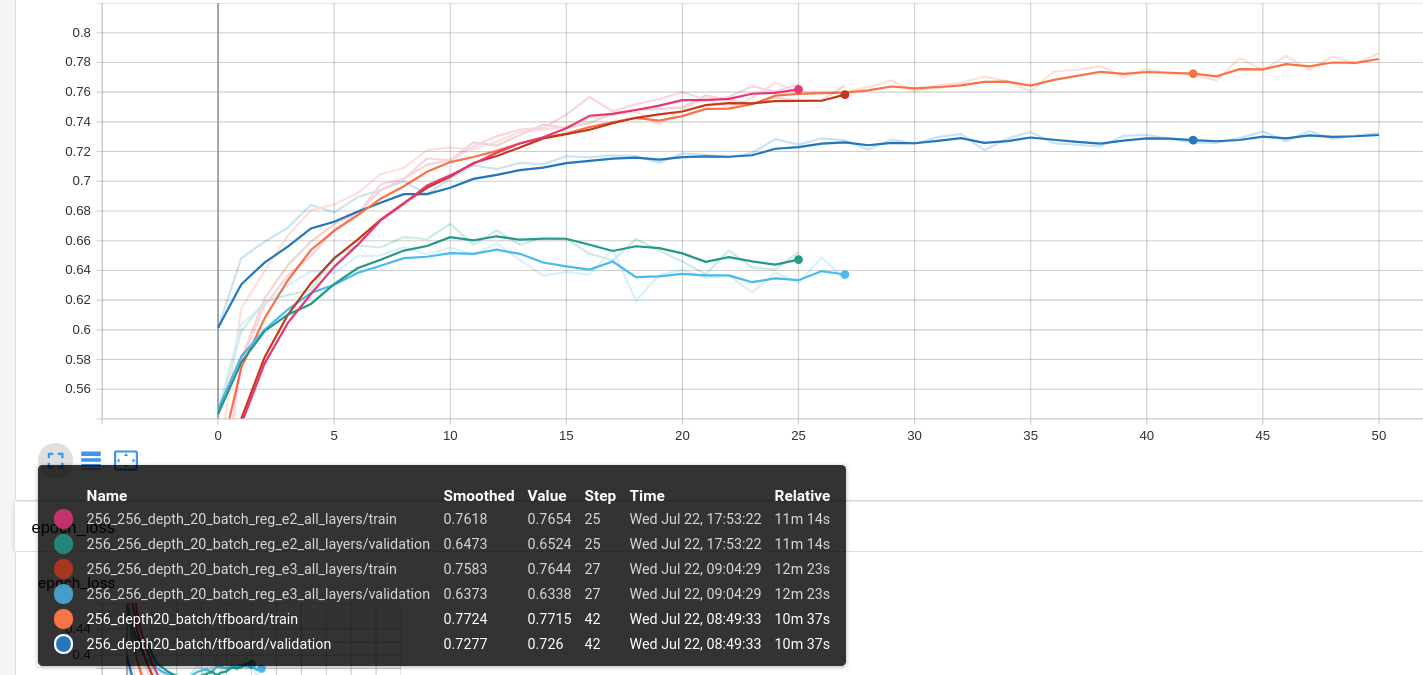


Más experimentos:

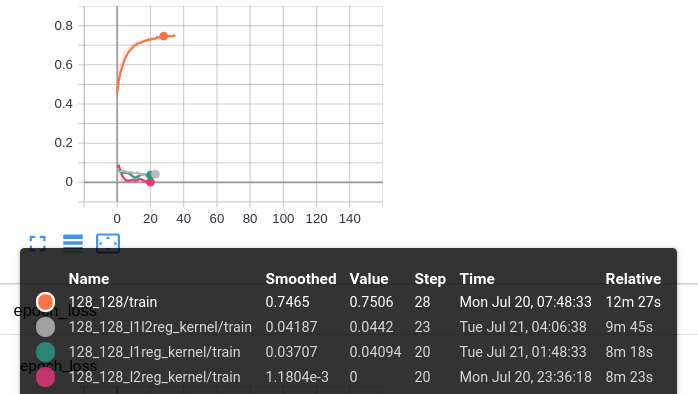
\* Capa con 512 y 256 con batch: se mejora respecto a 128 batch pero no demasiado: (0.74/0.72) < 256batch (0.74/0.73) < 256batch (0.76/0.73).

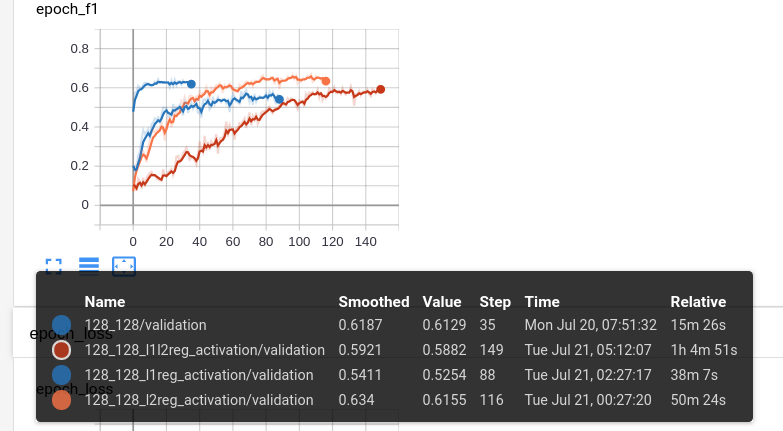
# Efecto del gradient clipping

# Efecto del número de capas

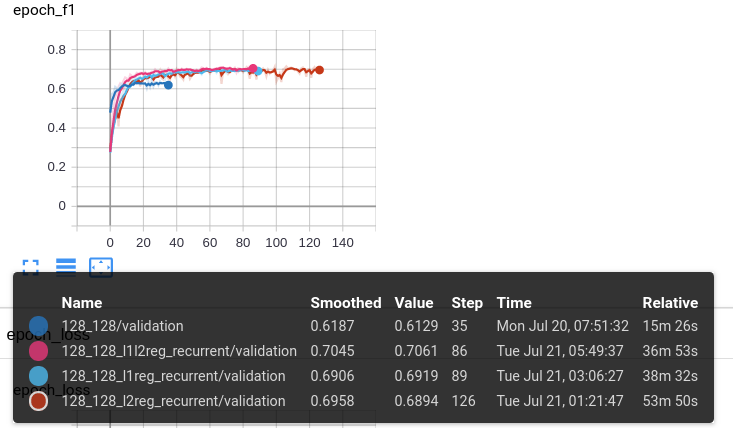
1. Añadiendo una segunda capa no se mejora, ni se mejora el entrenamiento ni en validación.
2. Hay que buscar explicación a esto, puede que las características que devuelve la capa no sean significativas.
3. Reducimos el tamaño de la segunda capa

# Efecto de la regularización en dos capas

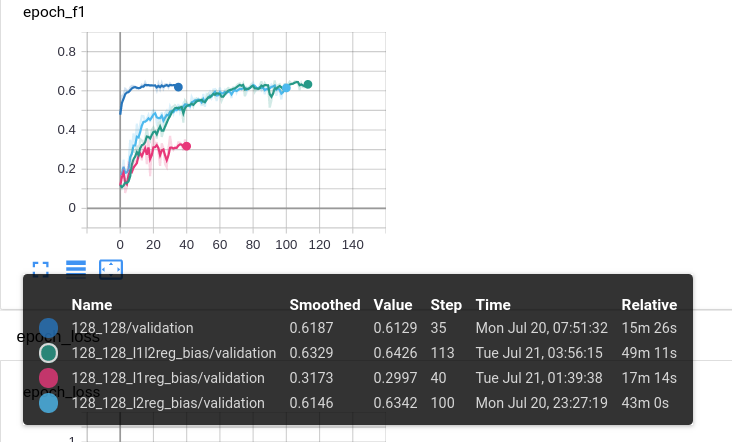
1. Las capas LSTM permiten cuatro tipos de regularizaciones, kernel, bias, activación y recurrencia, y para cada tipo se pueden aplicar tres normalizaciones, l1, l2 y elastica (combinada).
2. En todos los casos se ha aplicado regularización 0.001
3. La regularización **de kernel**, en los tres casos lleva a una pérdida de rendimiento drástica:
4. 
5. Regularizaciones de **activación**

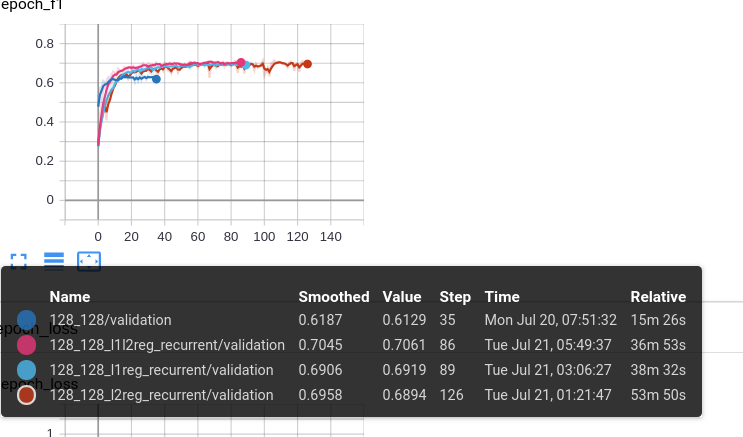


Regularizaciones **recurrentes**

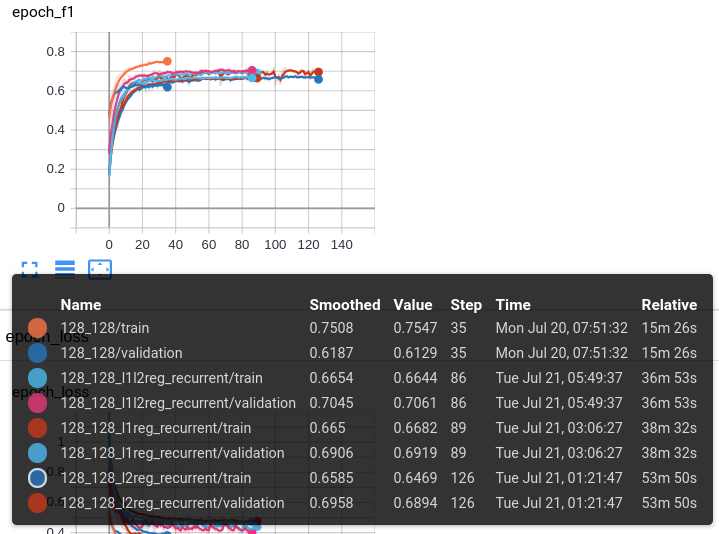


**Regularización de bias**

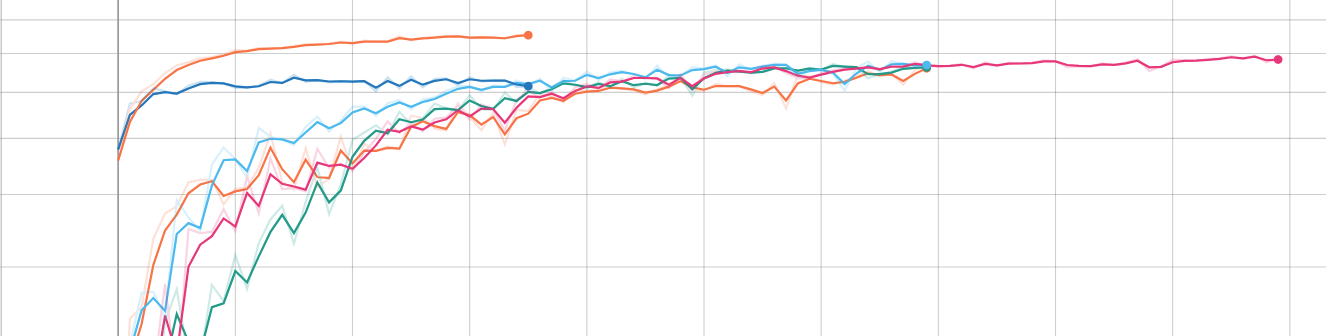




La regularización que mejor funciona es la recurrente con norma l2 o elastica (l1/l2), pero hay que ajustar el valor porque está provocando el efecto contrario, underfitting, la métrica de validación está por encima de la de entrenamiento. Hay que recudir el valor de regularización.



Utilizar r**egularización de recurrencia con valor 0.01 como máximo**, pero los tiempos de entrenamiento se alargan, pero la validación mejora.

Parámetro de regularización

Conclusiones:

* Aplicar regularización de kernel con valor máximo de 0.01, a partir de ese valor empieza a ocurrir el caso contrario, underfitting
* La regularización hay que aplicarla a todas las capas. Solo capa final (0.62,0.66) < Todas las capas (0.66,0.68)

Probar regularización 0.1 → Influye negativamente.

## Redes Conv1D

Mejor dos capas que una.

La capa dense intermedia no parece tener efecto, lo mismo que el flatten. Probar a mover a 128.

No se pasa del 0.7 ni en validación ni en entrenamiento.

**Conv1d32**

**Conv1d64**

Epoch 00082: val\_f1 did not improve from 0.70829

679/679 - 1s - loss: 0.2457 - f1: 0.7538 - val\_loss: 0.2984 - val\_f1: 0.7014

loss: 0.2737 - f1: 0.7257 - val\_loss: 0.2948 - val\_f1: 0.7050

**Conv1d32Conv64**

No tenemos suficiente longitud de secuencia para utilizar un maxpooling entre capas.

**LSTM and 1-D Convo**

**lutional Neural Networks for Predictive Monitoring of the Anaerobic Digestion Process**

|  |  |
| --- | --- |
|  |  |

[**https://sci-hub.tw/10.1007/978-3-030-30493-5**](https://sci-hub.tw/10.1007/978-3-030-30493-5)

# LSTM and Convolution Networks exploration for Parkinson’s Diagnosis

# Lstm, conv y conv → stack. El que mejor funciona CONV1D

# 

<https://ieeexplore.ieee.org/abstract/document/8809160>

**Deep learning based multi-temporal crop classification**

Conv1d y lstm para series temporales

<https://sci-hub.tw/https://doi.org/10.1016/j.rse.2018.11.032>

<https://towardsdatascience.com/lstm-fcn-for-cardiology-22af6bbfc27b>

Take aways:

* Dimension shuffling permite poder entrenar crear una rama de convolución.
* El stacking de LSTM y Conv no suele funcionar, ni por arriba ni por abajo es mejor crear dos ramas separadas, con embeddings distintos