Instituto Tecnológico y de Estudios Superiores de Monterrey

Departamento de Computación

Inteligencia artificial avanzada para la ciencia de datos I

Grupo 101

Módulo 2 Análisis y Reporte sobre el desempeño del modelo

Luis Ignacio Ferro Salinas

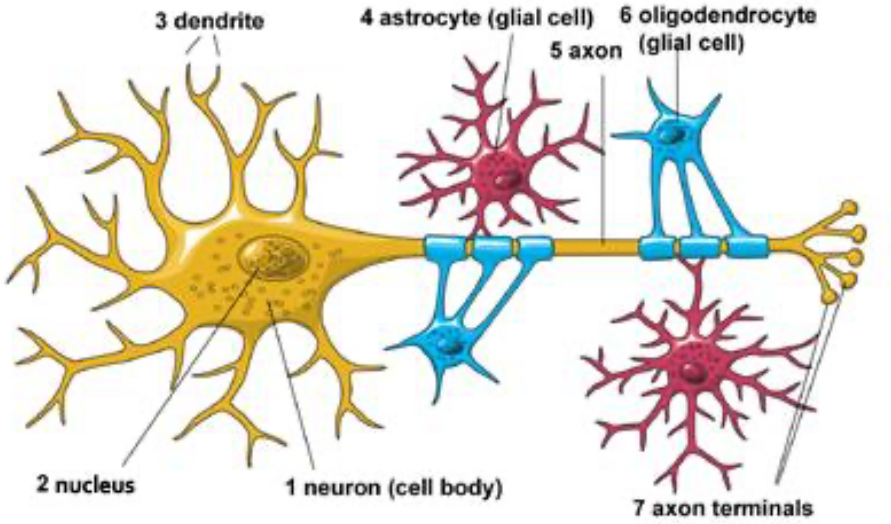
A01378248

13 de septiembre de 2022

Explicación de algoritmo de red neuronal.

El algoritmo que escogí utilizar es el de la red neuronal, este algoritmo me intriga porque se basa, de forma no tan exacta, en el funcionamiento de nuestro cerebro, en donde tenemos literalmente una telaraña de neuronas, que se interconectan formando una red.

Entonces el algoritmo se basa en el hecho de que una neurona n tiene un potencial de acción, que es un límite de voltaje que debe de recibir de todas las neuronas que pasan su voltaje a n, para que esta misma neurona n, pueda propagar el voltaje hacia su axón, que es la rama que se extiende de n para propagar el voltaje a otras neuronas.



Lo que ocurre en la red neuronal computacional es mucho menos complicado. Lo primero es la entrada, que en la mayoría de los casos es un vector de características, como en el problema que yo escogí del Titanic, el cual tiene las siguientes características para cada persona.

```
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  889 non-null    int64
 1   Survived     889 non-null    int64
 2   Pclass       889 non-null    int64
 3   Age          889 non-null    float64
 4   SibSp        889 non-null    int64
 5   Parch        889 non-null    int64
 6   Fare         889 non-null    float64
 7   Embarked     889 non-null    int64
 8   Sex_female   889 non-null    uint8
 9   Sex_male     889 non-null    uint8
```

Después si se codifican estas entradas como un vector, entonces se vería algo así:

$$\vec{I} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$$

Ahora, para esta entrada hay un vector de pesos, que se denota usualmente como $\vec{w}$

$$\vec{w} = (w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9)$$

De hecho usualmente hay una entrada constante x_0 que siempre es 1 para permitir un término de intercepto.

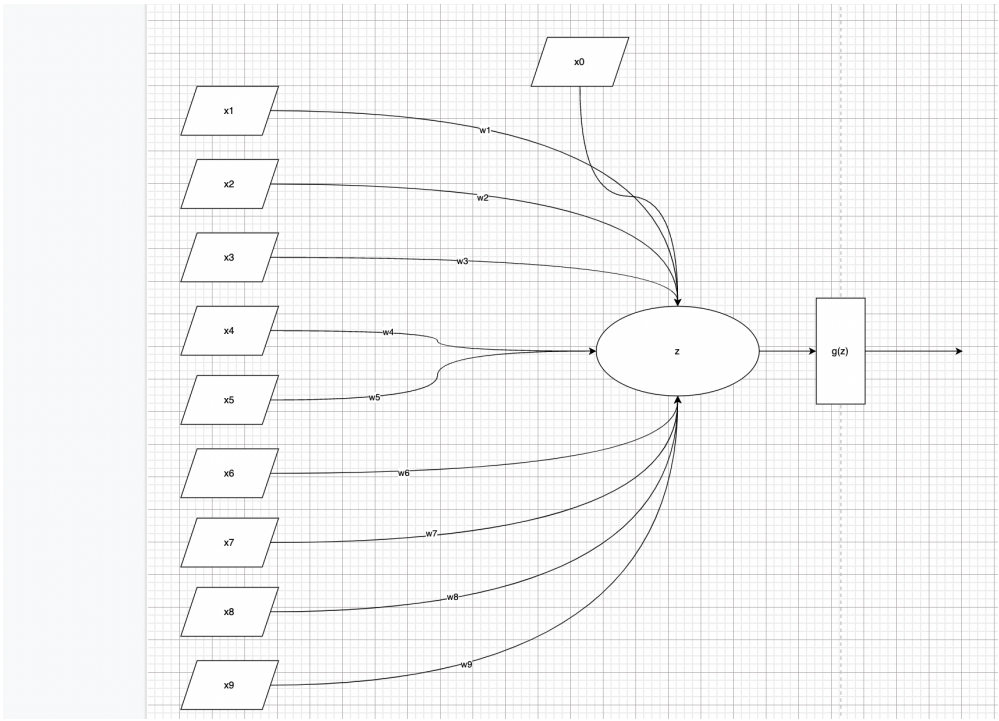
$$\vec{I} = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$$
$$\vec{w} = (w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9)$$

Entonces para pasar todas las entradas a la neurona n lo que se hace es reducir las entradas con una suma ponderada con los pesos $\vec{w}$, o un producto escalar entre $\vec{I}$ y $\vec{w}$.

$$z = \vec{I} \cdot \vec{w} = (x_0 w_0, x_1 w_1, x_2 w_2, x_3 w_3, x_4 w_4, x_5 w_5, x_6 w_6, x_7 w_7, x_2 w_8, x_9 w_9)$$

Para que el valor sea propague fuera de esta neurona n hacia cualquier otra neurona que esté después de esta, o para representar el resultado como una probabilidad, se puede pasar por una función de activación $g(z)$, este es uno de los hiperparámetros que configuro en mi actividad.

Entonces este escenario de una sola neurona se podría ver de la siguiente forma:



Para optimizar estos parámetros se usa el algoritmo del descenso del gradiente en su variante “ADAM”

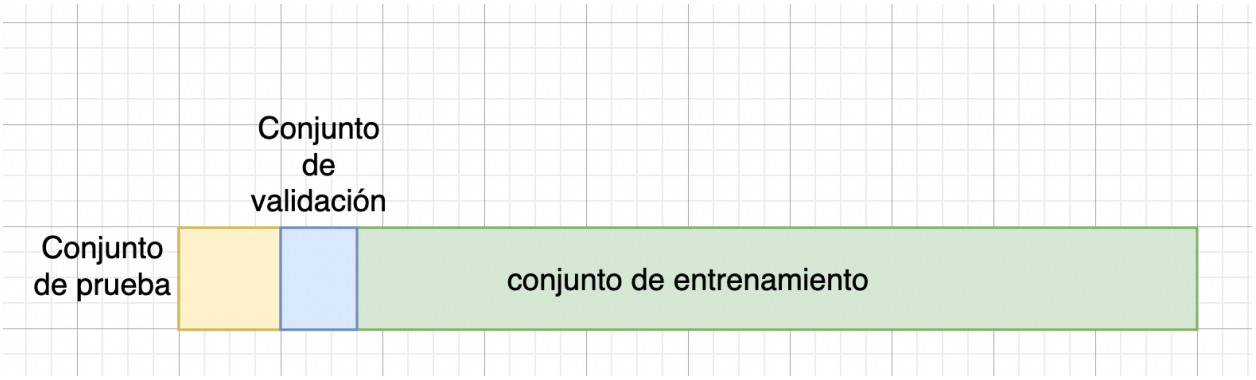
División de dataset

En mi implementación yo utilizo el conjunto con las características que ya describí, este conjunto tiene 889 muestras.

De este conjunto, hago una separación en subconjuntos de prueba, entrenamiento y validación.

Uso el 90% de las muestras para el entrenamiento y validación y el 10% para las pruebas. Del 90% de entrenamiento, utilizo el 10% de este subconjunto para la validación. En referencia al conjunto original, el conjunto de validación es el 9% de este conjunto original.

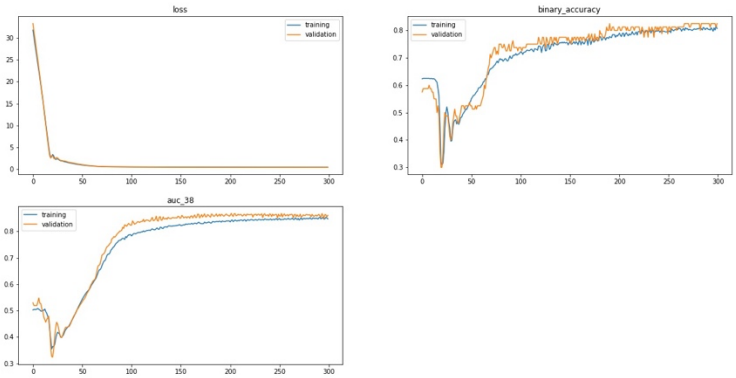
En proporción se vería de la siguiente manera:



Análisis del bias del modelo

Como métricas para medir la exactitud de mi modelo yo escogí primero la Accuracy, que me dice el porcentaje de predicciones correctas que hace mi modelo. También veo la AUC y la función de binary_cross_entropy.

En la versión entrenada, estas métricas se comportan de la siguiente manera con los datos de entrenamiento.

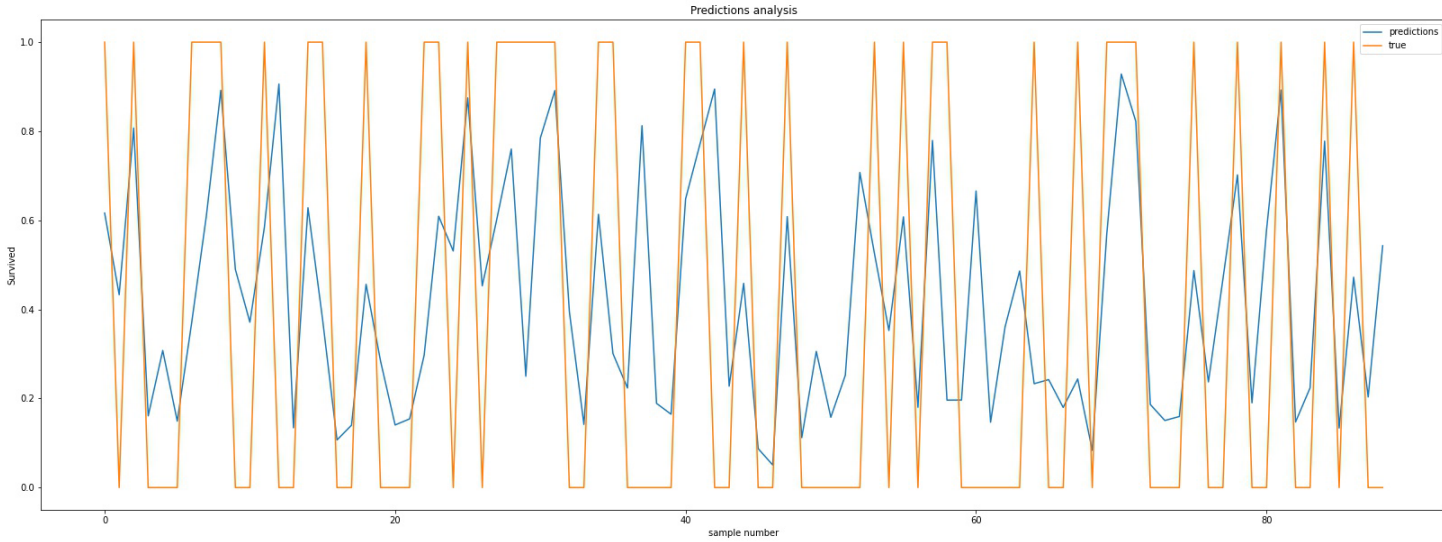


Mi accuracy al final era de 83%, lo que me dice que mi modelo predijo correctamente 8 de cada 10 veces de una forma simplificada. Creo que es un buen indicador del bias, no es remarcablemente alto, pero es bastante mejor que una predicción aleatoria.

También la función de pérdida, al final es de 0.41, que es una medida bastante baja y cercana a 0. Por estos indicadores digo que mi modelo tiene un bias moderadamente alto.

Análisis de la varianza del modelo

Para revisar la varianza de mi modelo voy a mostrar las predicciones que hace mi modelo cuando ya ha sido entrenado con los datos de pruebas.



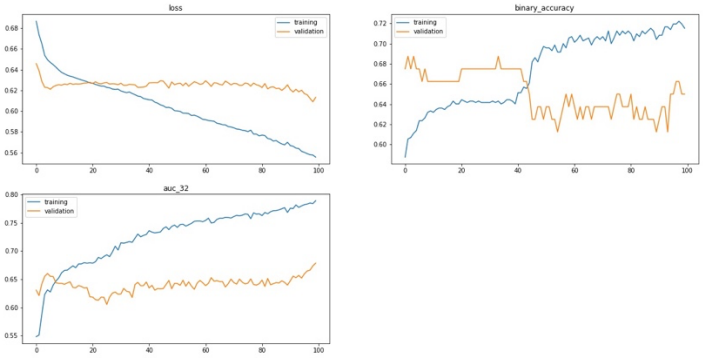
Como se puede ver, mi modelo no tiene tanta varianza cuando tiene que hacer predicciones porque cuando el valor es 1, las predicciones no se alejan tanto del valor real, en general están alrededor de 0.5 de distancia del valor real. De hecho esto también ocurre de una manera similar cuando el valor real es 0. Por esto creo que la varianza es baja.

Ya usando las métricas, por ejemplo, en mi conjunto de prueba la desviación estándar de los valores reales es 0.48625968831477334, mientras que la desviación estándar de las predicciones de mi modelo es 0.24844232

Esto expresa que hay más variación en los datos originales que en mis predicciones, por lo que mi modelo tiene una varianza tal vez baja.

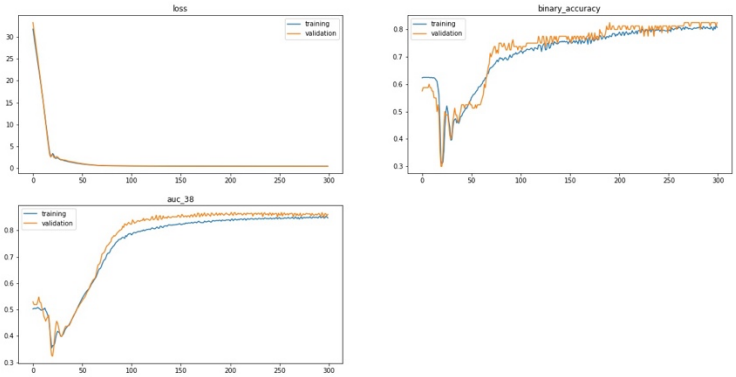
Análisis de overfitting y underfitting

De hecho durante el proceso de cambiar mis hiperparámetros, yo me encontré con una combinación en la que parecía que comenzaba a haber overfitting, como se puede ver a continuación:



En este caso las métricas para el entrenamiento seguían mejorando pero las métricas de validación comenzaban a empeorar, lo que indica el overfitting. En este caso particular era porque el número de muestras utilizadas para actualizar los parámetros eran muy pocas, entonces era posible que mi modelo estuviera favoreciendo los cambios solamente para estas pocas muestras pero estos cambios no reflejaban lo que necesitaba toda la población para mejorar globalmente.

Me dí cuenta de esto y evité esta combinación para que no hubiera overfitting. Finalmente las métricas siguen un progreso similar durante el entrenamiento, tanto para el conjunto de entrenamiento, como para el conjunto de validación, esto significa que no hay overfitting. Además, como las métricas parecen converger y no cambiar mucho después de ciertas epochs, significa que el modelo tiene desempeño bueno y tampoco hay un overfitting.



Ajustes para mejorar el modelo

Yo configuré varios parámetros para intentar mejorar mi modelo, los cuales eran:
Hiperparámetros:

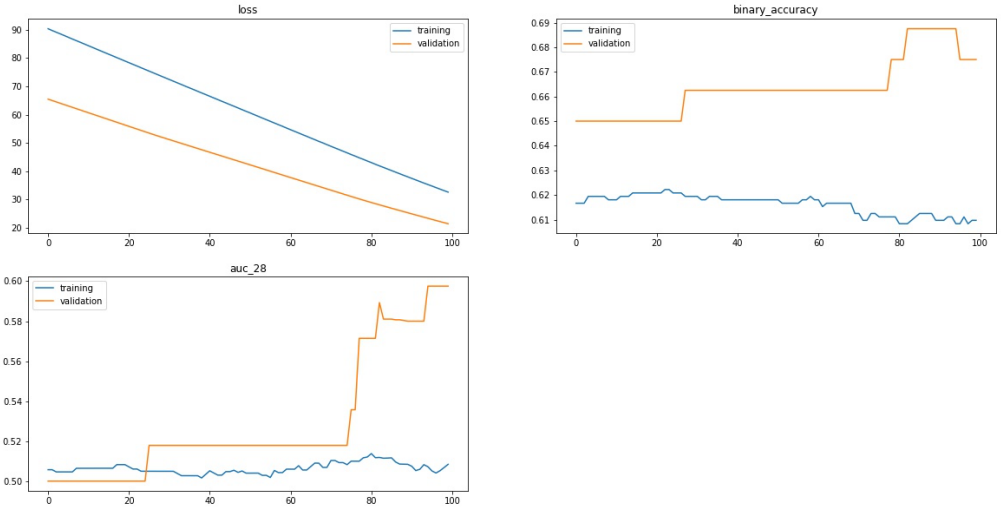
- Arquitectura del modelo (número de capas intermedias)
- Batch size
- Epochs
- Funciones de activación intermedias

Describo el proceso completo que seguí a detalle para mejorar mi modelo:

Arquitectura del modelo

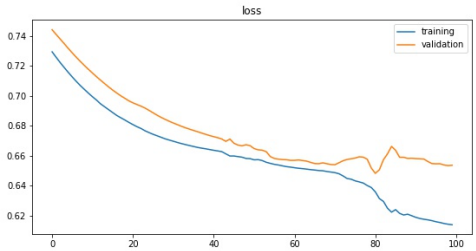
- batch_size=256,
- epochs=100,
- funciones de activación intermedias=sigmoid

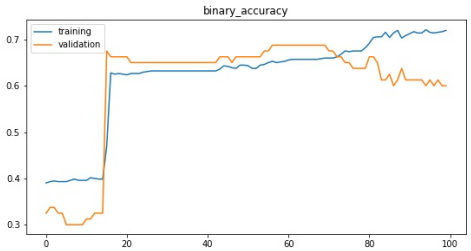
Una sola neurona (regresión logística) 9 x 1

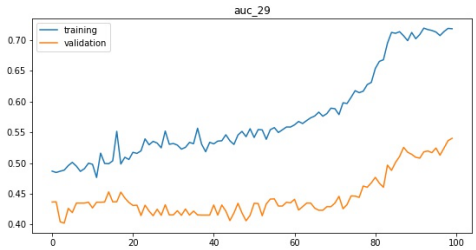


En este caso se ve que están un poco separadas las métricas de entrenamiento y de validación, pero parece que van en la misma dirección. Todas las métricas tienen mejora al final de las epochs contra al inicio.

Una capa intermedia 9 x 16 x 1

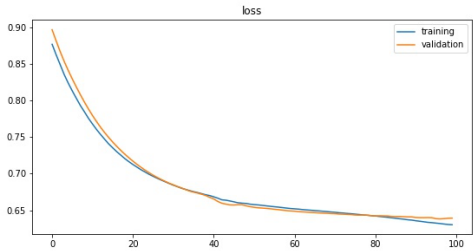


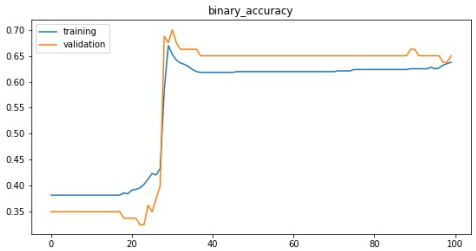


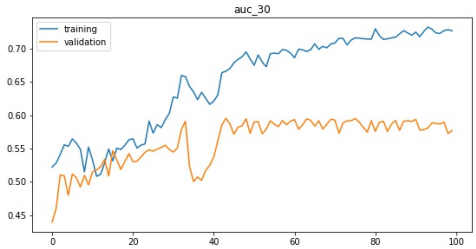


En este caso, las métricas de pérdida y de accuracy van bastante juntas, pero la AUC se separa bastante. Las métricas tienen mejores valores que en el caso anterior. Todas las métricas mejoran al final de las epochs contra el inicio.

2 capas intermedias 9 x 32 x 8 x 1





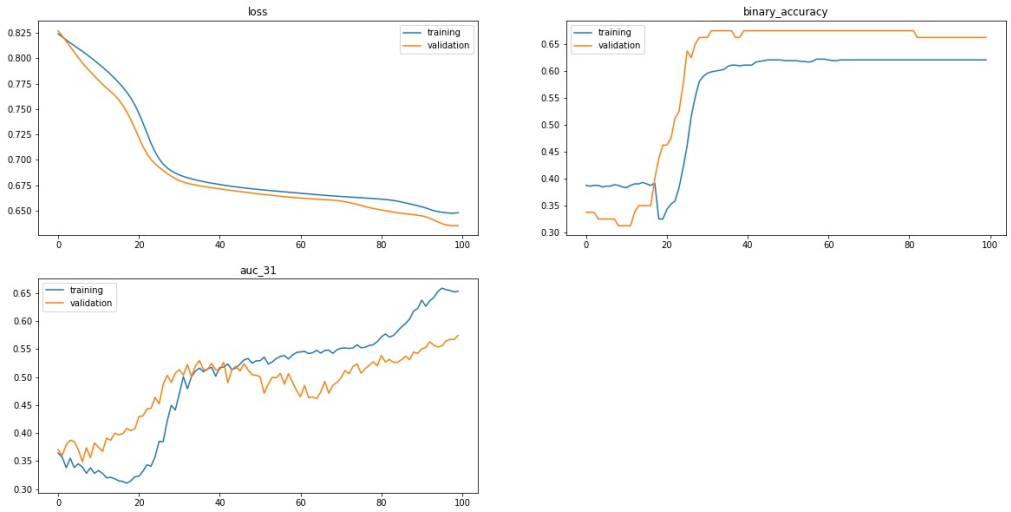


Parece que en este caso van bastante juntas las métricas de validación y training, excepto en la AUC. Las métricas tienen valores similiares contra la arquitectura anterior.

Batch_size

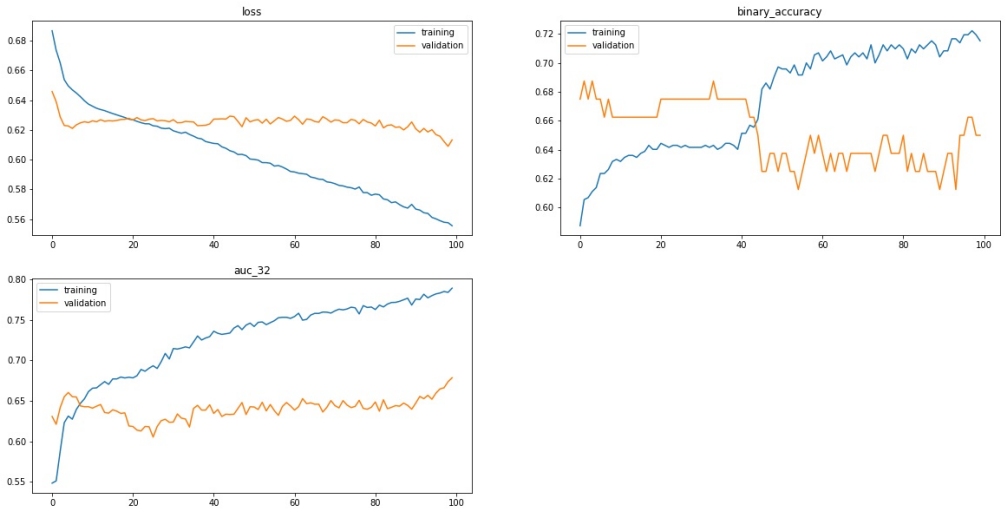
Arquitectura 9 x 16 x 1
Epochs 100
Funciones de activación intermedias sigmoid

Batch_size 800 (todas las muestras de entrenamiento)



Parece que la métrica de AUC se separa un poco entre training y validation, no hubo gran cambio en los valores de las métricas contra 256 muestras, ahora pruebo 64 muestras.

Batch_size 64

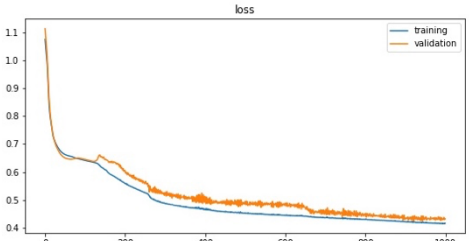


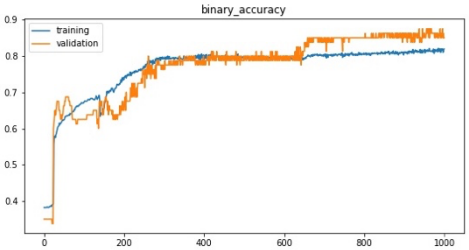
En este caso ya vemos bastante diferencia entre la forma en que cambian las métricas a través de las epochs entre validation y training, posiblemente 64 muestras en este caso no son suficientes para guiar el descenso del gradiente de la mejor manera.

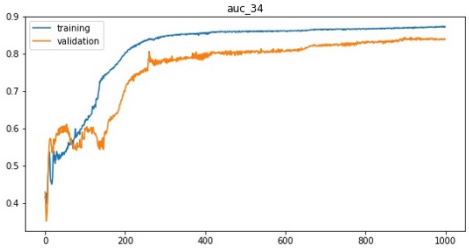
Epochs

Arquitectura del modelo 9 x 16 x 1
Batch size 256
Funciones de activación intermedias sigmoid

1000 epochs





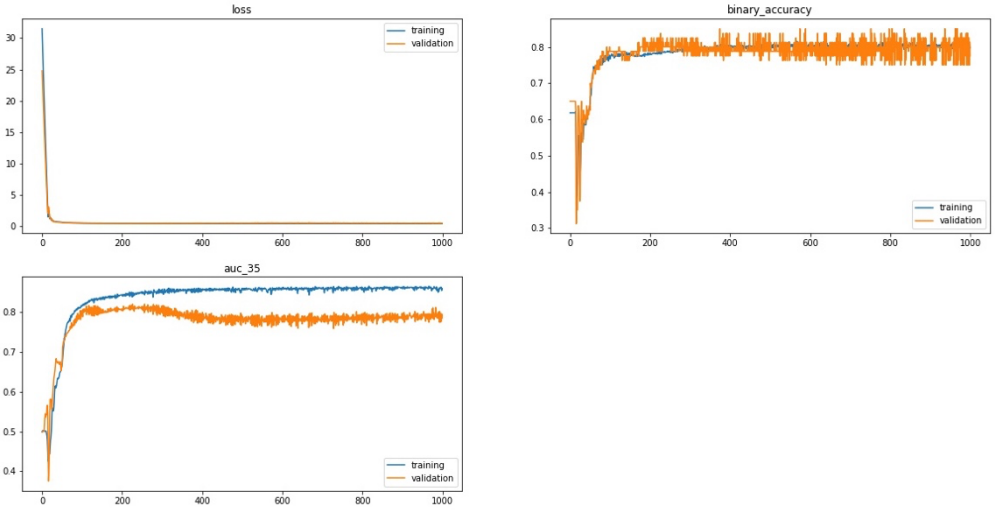


Estas métricas se ven bastante bien a través de las 1000 epochs, porque tenemos un crecimiento recíproco entre validation y training, ósea en general crecen las métricas en ambos casos de training y de validation. Parece también que se comienzan a estabilizar las métricas, ya no cambian tanto al final.

Funciones de activación intermedias

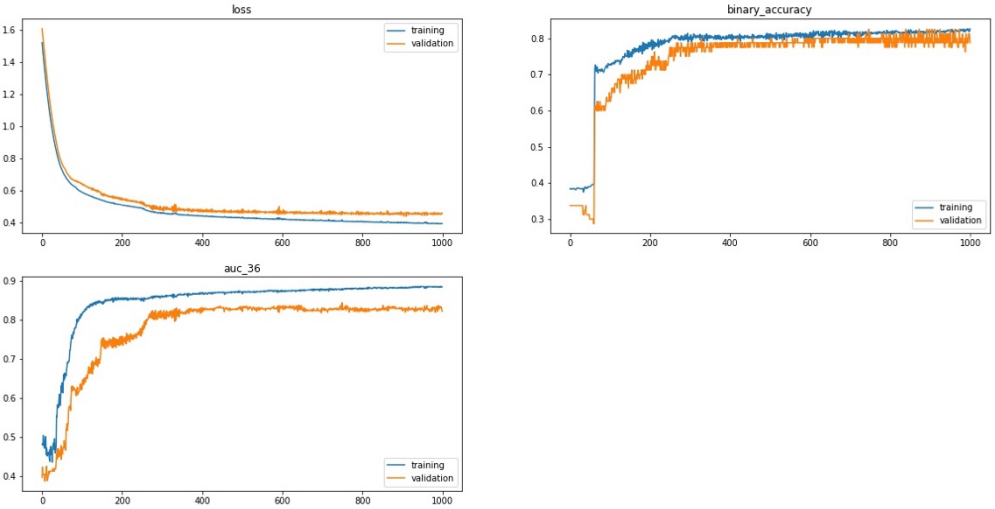
Arquitectura del modelo 9 x 16 x 1
Batch size 256
Epochs 1000

Funciones de activación intermedias ReLU



WOW, esto está interesante porque ahora la pérdida en el training y validation convirgió extremadamente más rápido que con la función intermedia de sigmoid.

Funciones de activación intermedia tanh

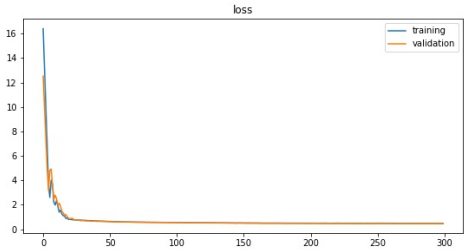


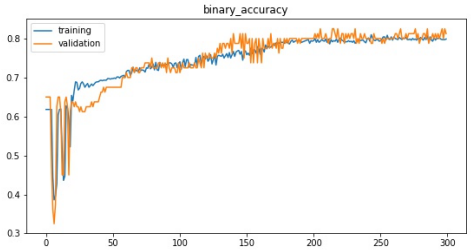
Es un resultado similar al de ReLU, pero creo que el comportamiento de ReLU es tan bueno en la loss que puedo bajar significativamente el número de epochs

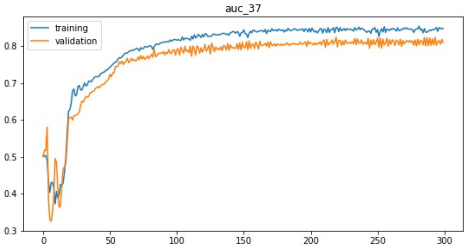
Configuración final

- Arquitectura del modelo 9 x 16 x 1
- Batch size 256
- Epochs 300

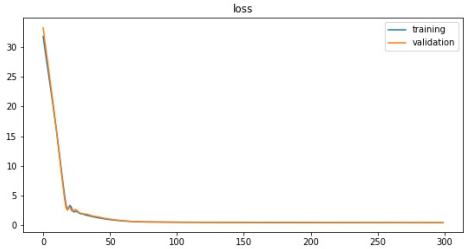
Funciones de activación intermedias ReLU

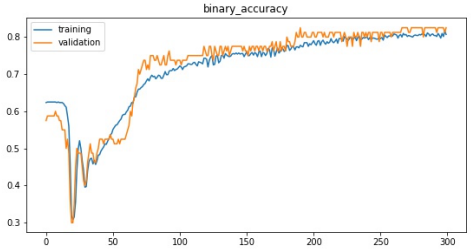


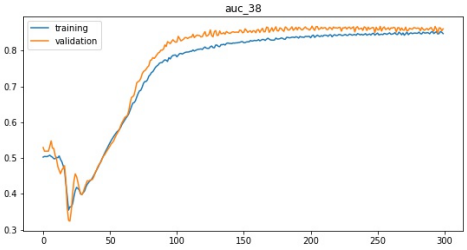




Ahora vuelvo a hacer el Split de training y testing con conjuntos distintos a ver si este comportamiento es reproducible







Y los resultados son bastante similares aún con conjuntos de training y testing diferentes.

Ahora vuelvo a hacer el proceso anterior pero ahora plasmando los valores finales de las métricas en tablas para visualizarlo en un espacio más reducido

Arquitectura	Perdida final	Perdida final validacion	Accuracy final	Accuracy final validación	AUC final	AUC final valifación
9x1	48.616893 76831055	45.798728 942871094	0.6138888 597488403	0.625	0.5272565 484046936	0.5026332 73601532
9x16x1	0.6185844 540596008	0.6147946 715354919	0.6277777 552604675	0.625	0.7423335 909843445	0.7297563 552856445
9x32x8x1	0.6177414 059638977	0.6066555 976867676	0.6277777 552604675	0.6374999 88079071	0.7405641 674995422	0.7679394 483566284

Batch size	Perdida final	Perdida final validacion	Accuracy final	Accuracy final validación	AUC final	AUC final valifación
800	.62526941 29943848	0.6237276 792526245	0.6347222 328186035	0.6499999 761581421	0.75624191 76101685	0.74358129 50134277
64	0.5504095 554351807	0.5367115 73600769	0.7180555 462837219	0.7749999 761581421	0.7886120 080947876	0.787030 93528747 56

Epochs	Perdida final	Perdida final validacion	Accuracy final	Accuracy final validación	AUC final	AUC final valifación
1000	0.3927774131298065	0.3862328529 35791	0.8277778029441833	0.875	0.3862328529 35791	0.8512178659439087

Activación de capa intermedia	Perdida final	Perdida final validacion	Accuracy final	Accuracy final validación	AUC final	AUC final valifación
ReLU	0.4429752 826690674	0.4084412 157535553	0.8013888 597488403	is 0.8500000 238418579	0.8548538 684844971	0.8459512 591362
Tanh	0.4262090 921401977 5	0.4198582 768440246 6	0.8180555 701255798	0.8625000 11920929	0.8606981 039047241	0.8439762 592315674

Configuración final	Perdida final	Perdida final validacion	Accuracy final	Accuracy final validación	AUC final	AUC final valifación
	0.4454258084297178	0.41798406839370728	0.78750002384185791	0.83749997615814209	0.85694807767868042	0.85352200269699097

Conclusión final

Se me hizo bastante útil hacer la división de entrenamiento y pruebas, porque de hecho pude entrenar varias ocasiones para verificar que mi modelo podía reproducir su accuracy con conjuntos ligeramente distintos. Además, el conjunto de validación fue muy bueno para darme cuenta cuando era momento de cambiar mi estrategia porque se generaba un overfitting.

Creo que la accuracy de mi modelo fue bastante optimizada, y si fuera a mejorar el bias de mi modelo sería porque se hace algún preprocesamiento adicional a los datos o feature engineering.

La varianza fue interesante porque me da a entender que el modelo intenta no tomar tantos riesgos al predecir para no tener errores tan grandes, y esto se ve reflejado en que la desviación estándar no era grande.