

## RECOMENDACIONES BASICAS PARA LA CONFIGURACION DE LAS REDES NEURONALES

### 1. Base de datos :

- a. Se debe trabajar con una base de datos. Utilizar la división de prueba de tren de scikit-learn para dividir los datos del tren obtenidos en 80% de tren y 20% de validación.
- b. Normalizar nuestra entrada para que los valores de entrada varíen de 0 a 1.
- c. Estandarizar / Normalizar las entradas puede hacer que el entrenamiento sea mucho más rápido y reducir las posibilidades de quedarse atascado en los óptimos locales, por lo tanto, es muy recomendable antes de ingresar los datos en la red para entrenar

### 2. Modelo Base

- a. Es una buena práctica comenzar con un modelo básico al principio, y luego seguir intentando mejorarlo en cada paso. Podemos comenzar con un modelo de línea de base de activación softmax.
- b. Para mejorar aún más el modelo, necesitamos saber cuáles son los hiperparámetros que podemos sintonizar en nuestra red densa, las siguientes secciones le darán una breve descripción de los hiperparámetros y cómo afectan el aprendizaje, pero antes de eso, comprendamos por qué necesitamos un ajuste de hiperparámetros
- c. Para encontrar el equilibrio correcto entre sesgo y varianza: - Es muy fácil lograr una precisión muy alta mientras entrenamos nuestros datos utilizando redes neuronales densas, pero estos podrían no generalizarse bien a nuestra validación y conjunto de pruebas también si tratamos de inhibirnos En arquitecturas profundas / complejas, siempre existe la posibilidad de tener una baja precisión en nuestros conjuntos de datos, por lo tanto, necesitamos encontrar el punto óptimo que se generalice bien y tenga una alta precisión. Cada hiperparámetro afecta la desviación de sesgo.
- d. Algunos modelos pueden ser víctimas de puntos de silla de montar y mínimos locales donde los gradientes son casi cero, por lo tanto, necesitamos ajustar los hiperparámetros como la velocidad de aprendizaje y cambiar el optimizador a Adam o RMSProp para no quedarse atascado y deja de aprender más.

### 3. Funciones de Activación

Problemas de la función de activación sigmoide y tanh: - Las funciones sigmoide y tanh están curvadas en sus extremos, lo que hace que los gradientes en esos puntos sean bastante bajos porque incluso cuando el

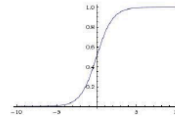
Referencia: <https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-i-hyper-parameter-8129009f131b>

valor antes de aplicar la activación sigmoidea aumenta considerablemente el valor después de aplicar por lo tanto, incluso el gradiente no aumenta mucho el impacto del aprendizaje. Pero Sigmoid y Tanh pueden ser buenas opciones para redes poco profundas (máximo 2 capas).

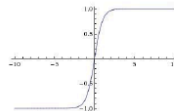
## Funciones de Activación

### Sigmoid

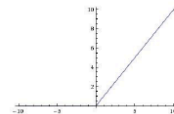
$$\sigma(x) = 1/(1 + e^{-x})$$



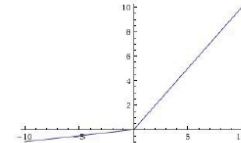
### tanh tanh(x)



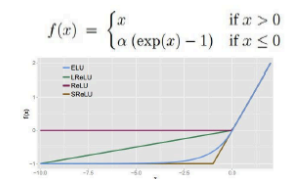
### ReLU max(0,x)



### Leaky ReLU max(0.1x, x)



### ELU



## 4. Configuración Recomendada

- Número de capas: debe elegirse con prudencia, ya que un número muy alto puede presentar problemas como un ajuste excesivo, desaparecer y explotar problemas de gradiente, y un número más bajo puede hacer que un modelo tenga un sesgo alto y un modelo de bajo potencial. Depende mucho del tamaño de los datos utilizados para la capacitación.
- Número de unidades ocultas por capa: -Estas también deben elegirse razonablemente para encontrar un punto óptimo entre el alto sesgo y la varianza. De nuevo depende del tamaño de los datos utilizados para el entrenamiento.
- Función de activación: - Las opciones populares en esto son ReLU, Sigmoid y Tanh (solo para redes poco profundas) y LeakyReLU. En general, elegir un ReLU / LeakyReLU funciona igual de bien. Sigmoid / Tanh puede funcionar bien para redes poco profundas. La identidad ayuda durante los problemas de regresión.
- Optimizador: es el algoritmo utilizado por el modelo para actualizar los pesos de cada capa después de cada iteración. Las opciones populares son SGD, RMSProp y Adam. SGD funciona bien para redes poco profundas, pero no puede escapar de los puntos de silla de montar y los mínimos locales en tales casos, RMSProp podría ser una mejor opción, AdaDelta / AdaGrad para datos dispersos, mientras que Adam es un favorito general y podría usarse para lograr una convergencia más rápida. Para mayor referencia <https://towardsdatascience.com/types->

Referencia: <https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-i-hyper-parameter-8129009f131b>

of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

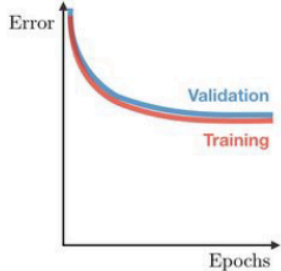
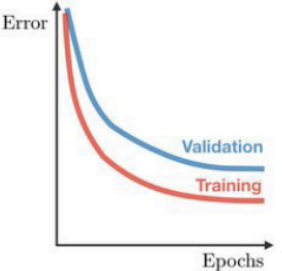
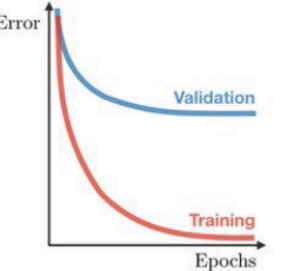
- e. Tasa de aprendizaje: es responsable de la característica central de aprendizaje y debe elegirse de tal manera que no sea demasiado alta en la que no podamos converger a mínimos y no demasiado baja para que no podamos acelerar el proceso de aprendizaje. Recomendado para probar en potencias de 10, específicamente 0.001, 0.01, 0.1, 1. El valor de la tasa de aprendizaje depende mucho del optimizador utilizado. Para SGD - 0.1 generalmente funciona bien, mientras que para Adam - 0.001 / 0.01, pero se recomienda probar siempre todos los valores del rango anterior. También puede usar el parámetro de decaimiento, para reducir su aprendizaje con varias iteraciones, para lograr la convergencia. En general, es mejor usar algoritmos de velocidad de aprendizaje adaptativo como Adam que usar una tasa de aprendizaje en decadencia.
- f. Inicialización: - No juega un papel muy importante ya que los valores predeterminados funcionan bien, pero aún así se prefiere usar la inicialización He-normal / uniforme mientras se usan ReLUs y Glorot-normal / uniform (el valor predeterminado es Glorot-uniform) para Sigmoid para obtener mejores resultados. Uno debe evitar el uso de cero o cualquier valor constante (igual en todas las unidades) inicialización de peso
- g. Tamaño de lote: es indicativo del número de patrones que se muestran a la red antes de actualizar la matriz de peso. Si el tamaño del lote es menor, los patrones serían menos repetitivos y, por lo tanto, los pesos estarían por todas partes y la convergencia sería difícil. Si el tamaño del lote es alto, el aprendizaje se volvería lento, ya que solo después de muchas iteraciones cambiará el tamaño del lote. Se recomienda probar tamaños de lote en potencias de 2 (para una mejor optimización de la memoria) en función del tamaño de los datos.
- h. Número de épocas: - El número de épocas es el número de veces que se muestran todos los datos de entrenamiento al modelo. Desempeña un papel importante en qué tan bien se ajusta el modelo en los datos del tren. Un gran número de épocas puede ajustarse en exceso a los datos y puede tener problemas de generalización en el conjunto de prueba y validación, también pueden causar problemas de gradiente de desaparición y explosión. Un número menor de épocas puede limitar el potencial del modelo. Pruebe diferentes valores en función del tiempo y los recursos computacionales que tenga.
- i. Abandono: (dropout) La probabilidad de mantenimiento de la capa de abandono puede considerarse hiperparámetro que podría actuar como un regularizador para ayudarnos a encontrar el punto óptimo de variación de sesgo. Lo hace eliminando ciertas conexiones en cada iteración, por lo tanto, las unidades ocultas no pueden depender mucho de ninguna característica en particular. Los valores que puede tomar

Referencia: <https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-i-hyper-parameter-8129009f131b>

pueden estar entre 0 y 1 y se basa únicamente en la cantidad de ajuste excesivo del modelo.

- j. Regularización L1 / L2: - Sirve como otro regularizador en el que los valores de peso muy altos se reducen para que el modelo no dependa de una sola característica. Esto generalmente reduce la varianza con una compensación de sesgo creciente, es decir, baja precisión. Se debe usar cuando el modelo continúa con un ajuste excesivo incluso después de aumentar considerablemente el valor de deserción (dropout)

## Regularización

	Underfitting	Just right	Overfitting
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none"> <li>• Complexify model</li> <li>• Add more features</li> <li>• Train longer</li> </ul>		<ul style="list-style-type: none"> <li>• Perform regularization</li> <li>• Get more data</li> </ul>