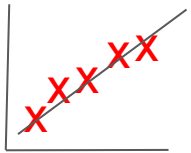
Consideraciones de un proyecto de Deep Learning

U-TAD

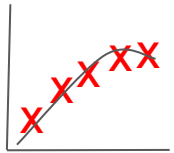
Overfitting & Underfitting (repaso)

- Si nuestra red neuronal tiene muchas capas y neuronas, es posible que la función hipótesis generada por el algoritmo se adapte muy bien al conjunto de datos de entrenamiento (J(W) = 0), pero falle al generalizar con nuevos ejemplos. Esto se conoce como **Overfitting**

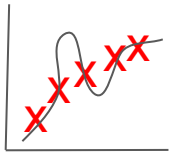
Underfitting



Óptimo



Overfitting



Objetivo: Error bajo para el conjunto de datos train, pero que generalice bien con val y test

$h(x)=w_0+w_1x_1$ $h(x)=w_0+w_1x_1+w_2x_1^2$ $h(x)=w_0+w_1x_1+w_2x_1^2+...+w_nx_1^n$

1 layer 2 layers n layers

Soluciones al overfitting

- Aumentar el conjunto de datos
- Reducción del número de capas
- Reducción del número de neuronas (mejor que reducir las capas)
- Regularización (lo vemos a continuación)
- Dropout (lo vemos a continuación)
- Data Augmentation

División del conjunto de datos (Train-Val-Test)

Evaluación de la hipótesis:

- Dividimos el conjunto de datos en **entrenamiento** y **pruebas**
- Se calcula la función hipótesis con el subconjunto de entrenamiento (se calculan los parámetros *W* minimizando el error de entrenamiento *J(W)*)
- Se evalúa la función hipótesis mediante el subconjunto de pruebas calculando su error

Necesitamos quedarnos con al menos el 90% para entrenamiento, ya que necesitamos conjuntos grandes (>1M de ejs) en Deep Learning (a diferencia de ML), y con un pequeño porcentaje para pruebas debería ser suficiente al disponer de muchos ejemplos en nuestro conjunto de datos.

Es normal que el error J sea menor en train que es test, pero deberían estar en un valor similar.

División del conjunto de datos (Train-Val-Test)

Ahora vamos a aplicar el concepto anterior a la **selección del modelo:**

- Seleccionamos el número de hidden layers óptimo:
  - Probamos con una, y si J_{train} es grande (underfitting), aumentamos a dos hidden layers
  - Si J_{train} es bajo y ligeramente inferior a J_{test}, habremos encontrado una buena solución
  - Si seguimos aumentando, hasta hacer una red neuronal muy compleja, podría tener un J_{train} muy bajo, pero un J_{test} alto (overfitting)

Cuando probamos el número de capas óptimo, podría ser también el número de neuronas óptimo, learning rate, función de activación, etc. Es decir, tenemos que evaluar todos los hiperparámetros con el conj. de datos de train y test. Y cuando lleguemos al modelo óptimo, habremos probado con varias arquitecturas de red neuronal diferentes, y puede llegar a ocurrir debido a estas diferentes pruebas con el mismo conjunto de test, sea óptimo únicamente para nuestro subconjunto de test. Es decir, que estemos provocando overfitting también en test, y continúe sin generalizar bien. Por lo que creamos otro subconjunto más (validación). Por ejemplo, train 98%, test 1%, val 1%

División del conjunto de datos (Train-Val-Test)

Selección de un modelo:

- Dividimos el conjunto de datos en **entrenamiento**, **validación** y **pruebas**
- Se calcula la función hipótesis con el subconjunto de entrenamiento (se calculan los parámetros W minimizando el error de entrenamiento *J(W)*)
- Se calcula el valor óptimo de los hiperparámetros mediante la evaluación de las hipótesis anteriores con el subconjunto de validación
- Se evalúa la función hipótesis resultante mediante el subconjunto de pruebas calculando su error

División del conjunto de datos (Train-Val-Test)

Selección de un modelo:

- Entreno con mi conjunto de datos de train, compruebo que no se produzca under o overfitting comparando J_{train} y J_{val}
- Voy aumentando las capas hasta que J_{train} y J_{val} sean bajos, entonces probaré con J_{test}. Y si el error aquí es alto, la arquitectura no es óptima (y tendremos que iterar sobre ella). Y si es bajo, entonces sí podremos afirmar que generaliza bien para ejemplos no vistos anteriormente.

¿Qué es la regularización?

Existen técnicas específicamente diseñadas para reducir el overfitting, como la regularización:

- La regularización agrega una penalización en los diferentes parámetros del modelo para reducir la libertad del modelo (J+penalización). Por tanto, es menos probable que el modelo se ajuste al ruido de los datos de entrenamiento y mejorará las capacidades de generalización del mismo
- Mantiene todas las características, pero reduce la magnitud de los parámetros *W*
- La regularización funciona bien cuando tenemos muchas características ligeramente útiles.

Regularización (L2 regularization)

$J(w, b) = 1/m * \Sigma L(\hat{y}_{(i)}, y_{(i)}) + \lambda / 2m * ||W||_2^2$ donde $||W||_2^2 = \Sigma W_j^2$

Gradient Descent tratará de minimizar la función de error, pero nunca podrá llegar a un límite de decisión (función de hipótesis), que se adapte exactamente al conjunto de datos de entrenamiento, ya que para ello debería tener un error cero respecto a ellos.

Al añadir esa penalización el error no será cero y continuará calculando modificando el valor de los parámetros hasta que se haga cero esta función de error con el término de penalización (o regularización) y, por tanto, llegaremos a un punto en el que nuestro modelo no se ajuste tan bien al conjunto de datos, pero sí se ajustará mejor a la tendencia global

Regularización en RNAs

$$J(w_{(1)}, b_{(1)}, w_{(2)}, b_{(2)}, \ldots, w_{(l)}, b_{(l)}) = 1/m \sum L(\hat{y}_{(i)}, y_{(i)}) + \lambda / 2m * \sum_{l=1}^{L} ||W_{(l)}||_F^2$$

hiperparámetro de regularización

donde $||W||_F^2 = \sum_{i=1}^{n(l-1)} \sum_{j=1}^{n(l)} W_{ij(l)}^2$ <- Frobenius Norm

Ejemplo F:

$$= w_{11(1)}^2 + w_{12(1)}^2 + w_{13(1)}^2 + w_{21(1)}^2 + w_{22(1)}^2 + w_{23(1)}^2 + \ldots$$

Gradient Descent y Regularización

for j en # mini-batches {

for i en # ejemplos en mini-batch {

$\hat{y}^{(i)}$ = Forward Propagation

J += $L(\hat{y}^{(i)}, y^{(i)})$

$dw^{[1](1)} = dJ / dw^{[1](1)}$ $dJ/dw^{[1](1)} = d/dw^{[1](1)} \left(1/m \sum L(\hat{y}^{(i)}, y^{(i)}) + \lambda/2m \sum ||W^{(l)}||_F^2\right)$

backpropagation + regularization

$= \text{backprop} + \lambda/m * w^{[1](1)}$

//A partir de aquí haríamos lo mismo:

Actualización del valor de los parámetros

Dropout Regularization

Vamos a comenzar aplicando esta técnica, asignando una probabilidad a cada una de las capas, excepto a la output layer. Por ejemplo:

- hidden layer 1: p = 0.4
- hidden layer 2: p = 0.5
- hidden layer 3: p = 0.3

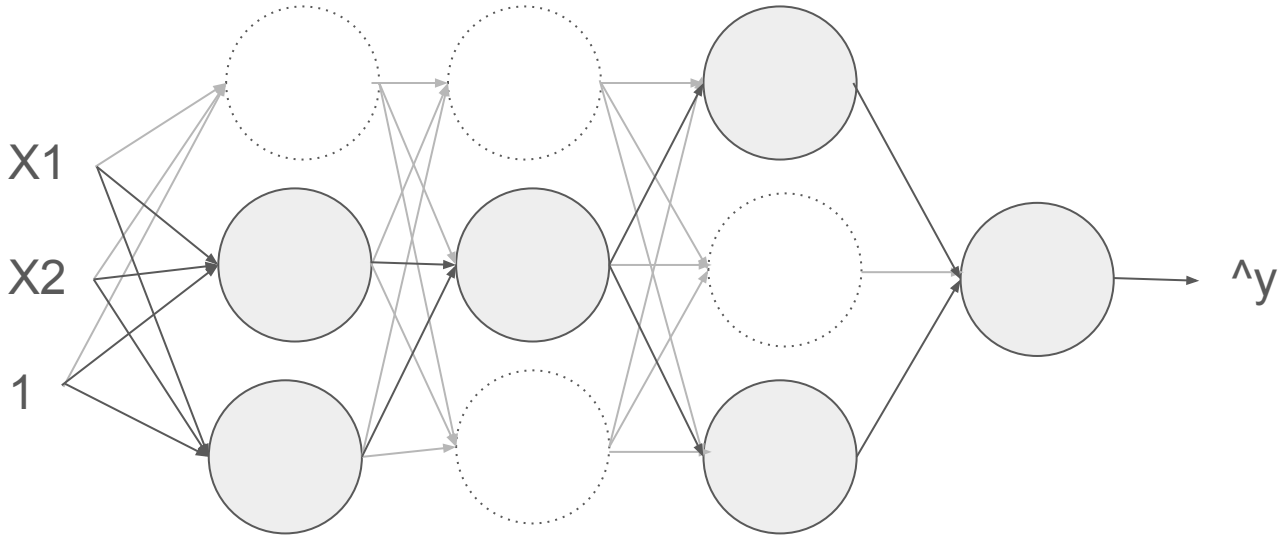
Estas p_s se van a corresponder con la probabilidad de que cada una de las neuronas de esa capa se mantengan en la siguiente iteración de entrenamiento con Gradient Descent. Es decir, veíamos que teníamos un bucle for por cada mini-batch y otro bucle for por cada ejemplo del minibatch, y dentro de este segundo bucle, teníamos los procesos de forward y backward propagation, calculando la derivada para saber en qué dirección debíamos modificar los parámentros. Y ya fuera del bucle los calculábamos.

Lo que se va a hacer, en cada forward y backward propagation, es ir eliminando neuronas de cada una de las capas en base al porcentaje que le hayamos indicado en cada capa concreta.

Por ejemplo, las neuronas de la segunda capa tendrán un 50% de probabilidad de desaparecer en la siguiente iteración con gradient descent

Dropout Regularization

En la práctica, básicamente, es que comenzaríamos entrenando con el primer ejemplo del mini-batch y lo que sucedería es que algunas de las neuronas en base a ese porcentaje que hemos asignado, desaparecerían. Y, además, desaparecen todas las conexiones tanto entrantes como salientes.



En las siguientes iteraciones (siguientes epochs), se activarán y desactivarán otras distintas

Dropout Regularization

Lo que se está haciendo es provocar que las neuronas de las diferentes capas no dependan de las mismas neuronas de la capa anterior (de las mismas input features que le llegan de las capas anteriores) cuando procesan sus características.

Esto hace que las neuronas “se especialicen menos” a la hora de identificar determinadas características y, por tanto, generalicen mejor para ejemplos que no están en nuestro conjunto de datos porque no se van a centrar en características muy específicas que tienen los ejemplos de nuestro conjunto de datos.

Esta técnica funciona muy bien, en concreto las redes convolucionales para detección de imágenes se usa con porcentajes altos, del 40 ó 50%. Y en las redes recurrentes, más orientadas a la detección o clasificación de texto, también utilizan dropout aunque en un porcentaje algo menor (20 ó 30%) pero que sigue siendo útil.

Si nuestra arquitectura estuviese provocando overfitting incluso aplicando regularización, podemos usar dropout, aumentando la probabilidad de dropout para tratar de reducir ese overfitting.

Otros mecanismos de regularización

Data augmentation

- Se aumenta el conjunto de datos de entrenamiento mediante la modificación de los ejemplos existentes

Por ejemplo, si nuestro problema consiste en identificar si en una imagen aparece un coche o no, entonces en nuestro conjunto de datos tendremos imágenes en las que aparecen coches etiquetadas con un 1, e imágenes en las que no aparecen coches etiquetadas con un 0. Lo que podemos hacer si necesitamos aumentar el conjunto de datos y no tenemos acceso a otras imágenes de coches, es coger las que ya tenemos y aplicarles transformaciones (voltear, recortar, etc)

Otros mecanismos de regularización

Early stopping

Consiste en detener el entrenamiento de nuestra red neuronal antes de que llegue al final, es decir, antes de que minimice esa función de error para el subconjunto de datos de entrenamiento.

Si llegada una cierta epoch, el error de train sigue bajando pero el error de val se estabiliza o incluso sube, se estaría produciendo overfitting. Es decir, tendríamos un error muy bajo para el subconjunto de datos de entrenamiento, pero en cuanto se prueba con ejemplos que no se han utilizado para entrenar se obtiene un error grande. Por lo que si observamos la gráfica, y vemos que a partir de un epoch x, el error de val aumenta, lo que haríamos es parar el entrenamiento en esa epoch, impidiendo que siga minimizando la función de error con los datos de train.

No es tan recomendable como las dos primeras que vimos, L2 o F regularization y dropout. Pero si no funcionan las técnicas anteriores, podemos tratar de usar esta.

Normalización de las entradas en RNAs

Normalización:

1. Restar la media:

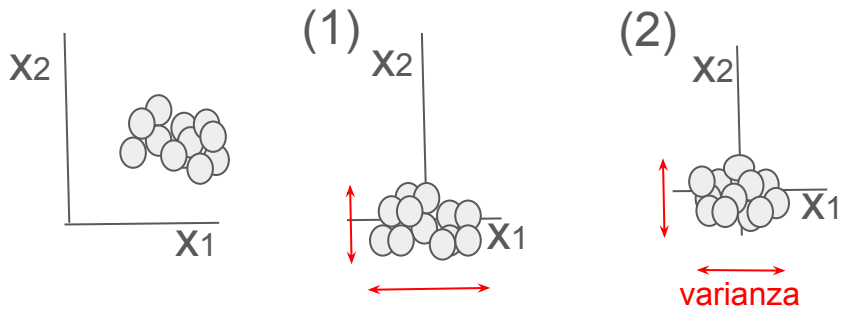
$$\bar{X} = 1/m \sum x_{(i)}$$

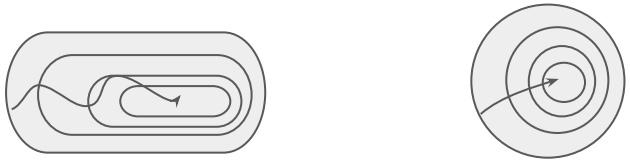
$$\mathbf{x} = x - \bar{X}$$

2. Normalizar la varianza

$$S^2 = 1/m \sum \mathbf{x}_{(i)}\text{^}2$$

$$\mathbf{x} = x / S^2$$





Normalizando, llegamos más rápido a ese valor de error mínimo global

Usaremos los mismos valores de media y varianza para los datos de val y test

Práctica: Clasificación de Tweets

Clasificación de Tweets de desastres naturales.ipynb

Construye un modelo de deep learning que realice predicciones sobre qué Tweets tratan de desastres reales y cuáles no.

https://www.kaggle.com/datasets/vstepanenko/disaster-tweets