Redes Neuronales

Redes neuronales

Perceptrón: el modelo de 1 sola neurona

Este es el modelo idealizado (si haces un modelo ya está idealizado)

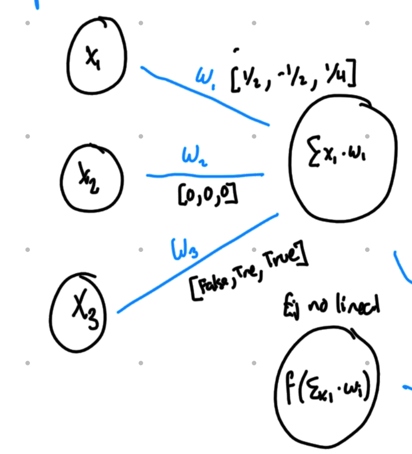
Vectores
- → matemáticos → 1,2,3,4 y 6,7,9,9 = 6+14+21+36 = 1.8 [illegible]
- → compu → $\begin{bmatrix}1\\2\\3\\4\end{bmatrix}\begin{bmatrix}6\\7\\7\\9\end{bmatrix}$ = 6, 14, 21, 36 .

son los que nos interesa pq la salida buscamos ej virginica, versicolor y setosa

Inputs
Capa de entrada

Neurona
capa oculta

Outputs
Capas de salida



$x_1 w_1 = [8\cdot 1/2,\ (-2)(-\frac{1}{2}),\ 4(\frac{1}{4})]$

$x_2 w_2 = [-6\cdot 0,\ 11\cdot 0,\ 17\cdot 0]$

$x_3 \cdot w_3 = [0, 0, \text{arroca}]$

$\sum x_i w_i = [8\cdot 1/2 + -6\cdot 0 + 0,\ -2\cdot -\frac{1}{2} + 11\cdot 0 + 0,\ 4\cdot 1/4 + 17\cdot 0 + \text{arroca}]$

$\hat{y}$

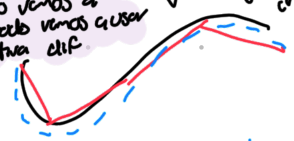
Aquí ya se genera setosa, virginica, versicolor (f.n. lineal)

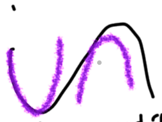
En escencia todo se puede linealizar no importa que tan complejo sea el comportamiento de la función

Solo que el costo computacional tmb crece

Funciones de activación

nosotros no vamos a linealizar todo vamos a usar una alternativa dif





ej una cuadrática

Combinación lineal: suma y punto escalar, solo te genera líneas rectas, combinas datos de entrada por productos constantes

mate ↑ (suma y punto escalar)

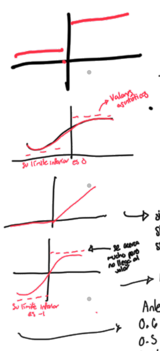
geométrico ↑ (líneas rectas)

Manipulación vs limpieza

Limpieza (ADE) análisis datos exploratorios → tenemos Na y vemos si los quitamos o los reemplazamos por algo

Manipulación → mueves # para tu conveniencia

Funciones de activación



- Función escalón: antes del 0 vale 0, después del 0 vale 1 (tiene o no tiene cáncer)
- Función sigmoide: nos da valores de 0 al 1, se usan en problemas de probabilidad en binario (+ q es o no es cáncer)
- Función ReLu (rectificador lineal unitario) → si el valor es − le asignas 0, si es + de 0 le asignas el valor de x, sirve para ayudar a discernir entre opciones → al normalizar mis testimos un rango −1 a 1 entonces todas las manda a setosa
- Función atan → lo mismo que la sigmoide pero puedes usar valores negativos
- Función softmax * pendiente

Antes	después (prob)
0.6	6/15
0.5	5/15
0.4	4/15

Funciona cuando tienes múltiples neuronas

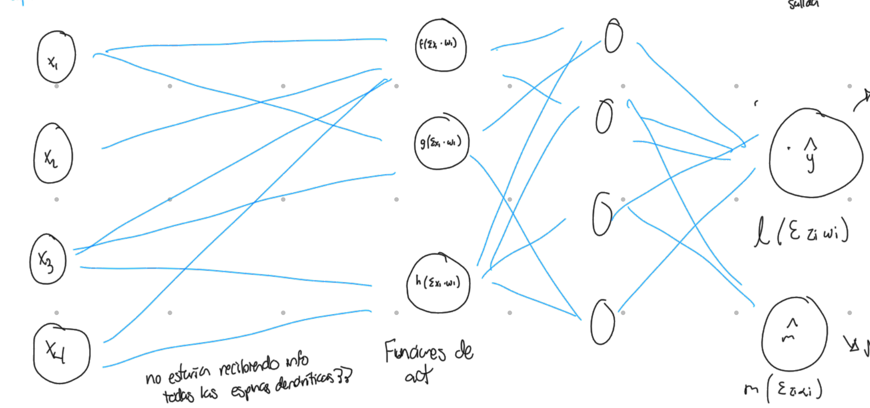
Inputs
Capa de entrada

Neurona
Capa oculta . . . Capa oculta m (podemos + # capas)

puedes definir # capas y cuantas neuronas quieres

Capa salida
No necesariamente tiene que ser 1 sola salida

La red neuronal está calculando los dif pesos para que nos de la salida



no estaría recibiendo info todas las esporas derechas??

Funciones de act

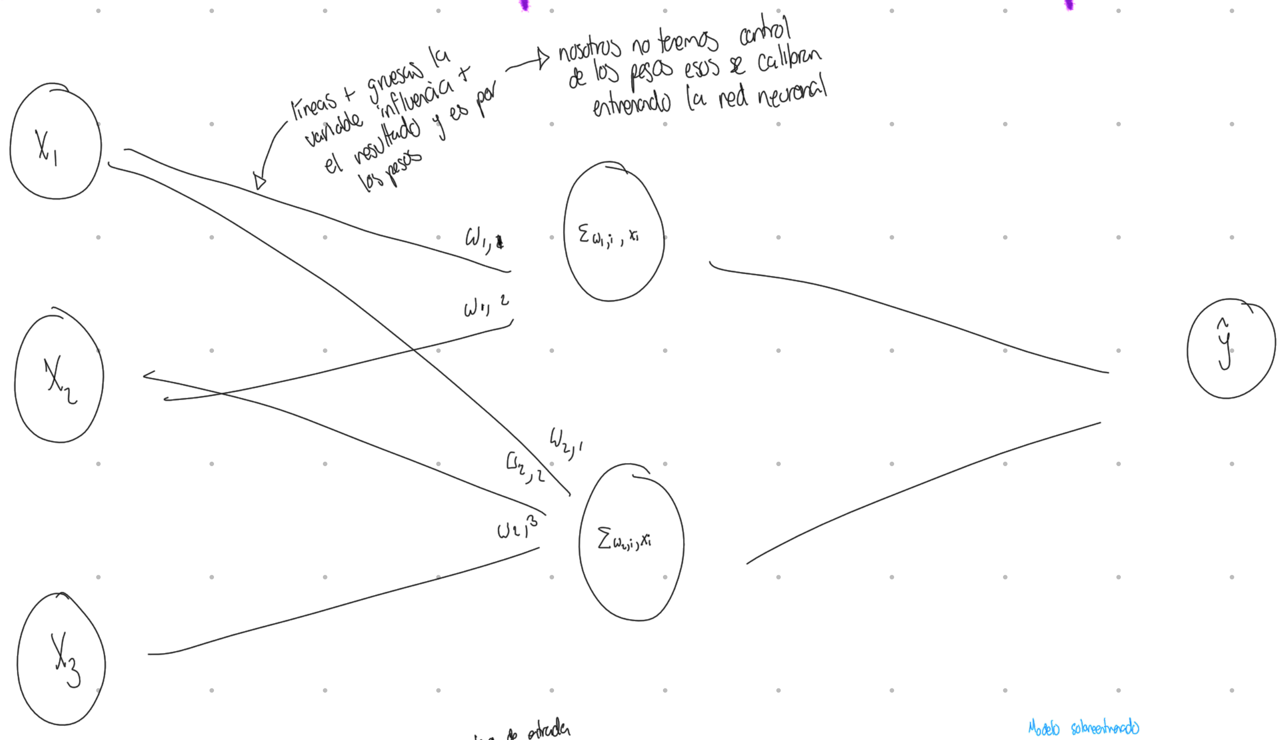
→ Virginica

→ Versicolor → si no es ninguna indirectamente te dice que es setosa

tenemos múltiples capas de salida pq tenemos dif clases (ej setosa, virginica, versicolor)

Capa de entrada Capa oculta Capa de salida

tu puedes crear + variables de entrada (al cuadrado multiplicarlas y así sobre las que ya existen)



vectores de entrada

$\underset{1 \times 3}{\Theta} \ \underset{3 \times 1}{\bar{X}} = \underset{1 \times 1}{\hat{y}}$ ← vector de salida

matriz omega es lo que tiene todos los pesos de nuestra red neuronal → lit son los pesos la ω

Modelo sobreentrenado

Test loss → el error se va a 0

Training loss → el error se dispara

Modelo subentrenado

Error grande en ambos conjuntos

Modelo bien

Cuando ambos valores disminuyen

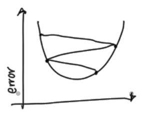
Entrenamiento → La recomendación de los pesos son valores entre 0 y 1, normalmente 1 pq así todas las neuronas pueden influir de la misma forma el resultado, ya después los cambias

- Forward propagation → propagación anterógrada → variables de entrada con pesos para encontrar el cálculo del error $e = \frac{1}{n}\Sigma(y-\hat{y})$ el error puede ser muy grande o pequeño pq estamos iniciando el entrenamiento de la red neuronal

↳ f(x) → máximos y mínimos usamos f'(x) usas algo de Taylor

$f(x) = \frac{f(x+h) + f(x)}{h}$ → encontrar el comp de futuros pesos de la función a partir de la derivada

- Backward propagation ← propagación retrógrada + método de Adams para encontrar el error y creo que luego averiguas pesos



Con la función de Adams calculas los futuros pesos con backward y forward 1 y 1 el punto es que entre más iteraciones tu esperarías llegar al mínimo error

Se le conoce como epoch

back → ford

back ↓ error y nuevos pesos

ford ↓

En algún punto se estanca el error

La h se usa para darle un peso al gradiente y acompaña a la derivada

La h es el impulso que le das → la velocidad a la que aprende la red neuronal

h bajo → el error se reduce lento

h alto → el error se reduce rápido pero no se recomienda pq si tu base es muy variable el comportamiento es errático (brinca como loko)

Valores recomendados 1, .01, .001

Redes neuronales

— Nosotros sí podemos controlar epoch y funciones de activación

Función del error o costo

$\hat{y} - y$

modelo groundthrut

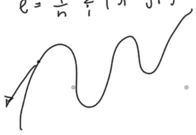
→ es cuadrática → la función es cóncava → cuando queremos minimizar el error si encontremos un mínimo → tenemos siempre +

$$\hat{e} = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

así se comporta el error pero solo para 1 perceptrón si tienes + neuronas valió

→ la función de costos podría verse así

$$\hat{e} = \frac{1}{n}\sum_{i}(y_i - \hat{y}_i)^S$$



$\hat{e}$ mínimo local vs mínimo global

se estanca aquí en el mínimo local