

Redes Neuronales 2.0

Fernanda Sobrino

6/21/2021

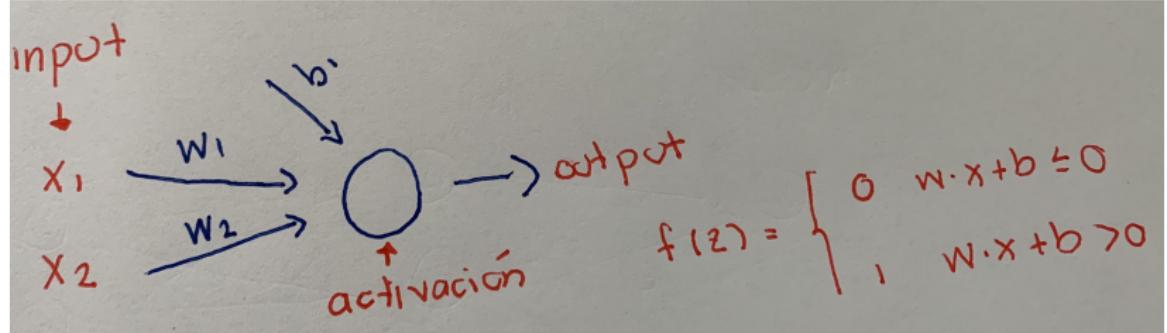
Perceptron

Ejemplo Backpropagation

Cómo escribimos esto en R

Perceptron

Representar la función lógica OR con una red neuronal



Función lógica OR

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	1

Función lógica OR

$$w_1 = .5 = w_2 \quad b_1 = -.5$$

* Si $x_1 = 0 = x_2$ queremos 0

$$0 \cdot .5 + 0 \cdot .5 - .5 = -.5 \leq 0 \Rightarrow 0$$

* Si $x_i = 0, x_j = 1$ queremos 1 → Hay que actualizar las w 's

$$0 \cdot .5 + 1 \cdot .5 - .5 = 0 \leq 0 \Rightarrow 0$$

Actualizamos los pesos

- ▶ actualizar pesos usando:
 - ▶ tasa de aprendizaje: $\eta = 0.5$
 - ▶ $\epsilon = \text{actual} - \text{prediction} = 1 - 0 = 1$
 - ▶ nuevo peso dado por $w'_i = w_i + \eta * \epsilon$
 - ▶ $w_i = 0.5 + 0.5 * (1) = 1$
- ▶ $\Rightarrow x_i = 1, x_j = 0$ ahora $x \cdot w + b = .5 > 0 \Rightarrow \text{output} = 1$
Correcto
- ▶ funciona en los otros dos casos?
 - ▶ $x_i = 1, x \cdot w + b = 1.5 > 0 \Rightarrow 1$ Correcto
 - ▶ $x_i = 0, x \cdot w + b = -0.5 \leq 0 \Rightarrow 0$ Correcto

Por qué necesitamos redes de más de un nodo y más de una capa?

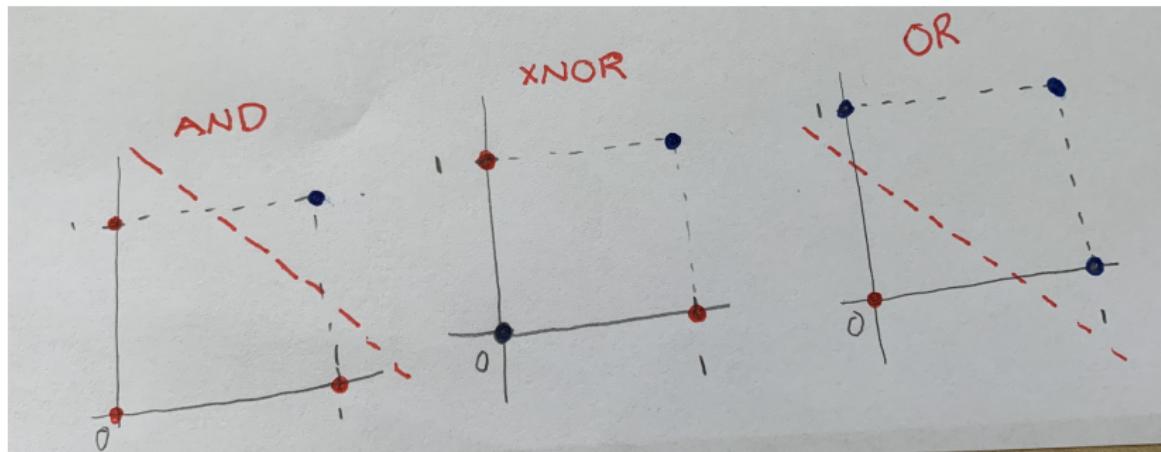
Problema, encontrar un red neuronal que describa lo siguiente:

X_1	X_2	Y
0	0	1
0	1	0
1	0	0
1	1	1

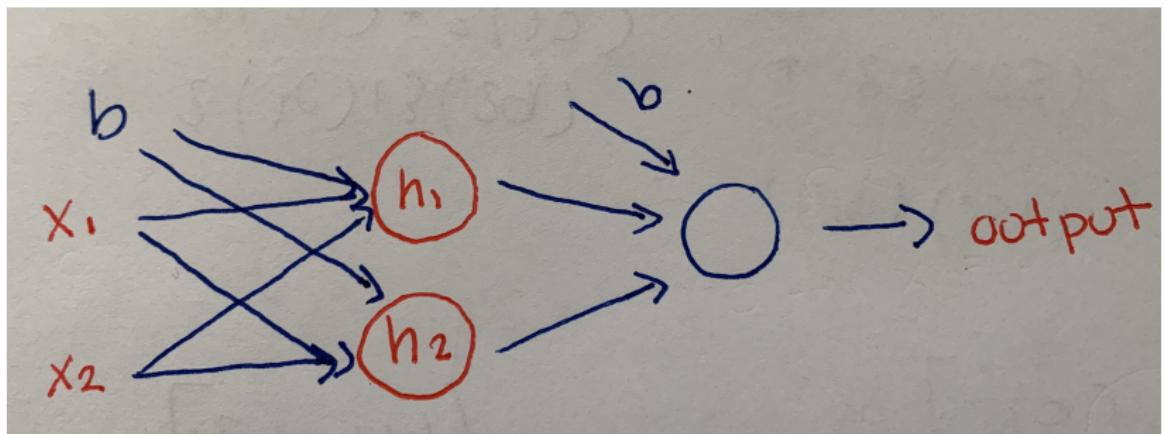
Por qué necesitamos redes de más de un nodo y más de una capa?

- ▶ No va a existir una red de un solo nodo capaz de describir lo anterior
- ▶ Por qué?

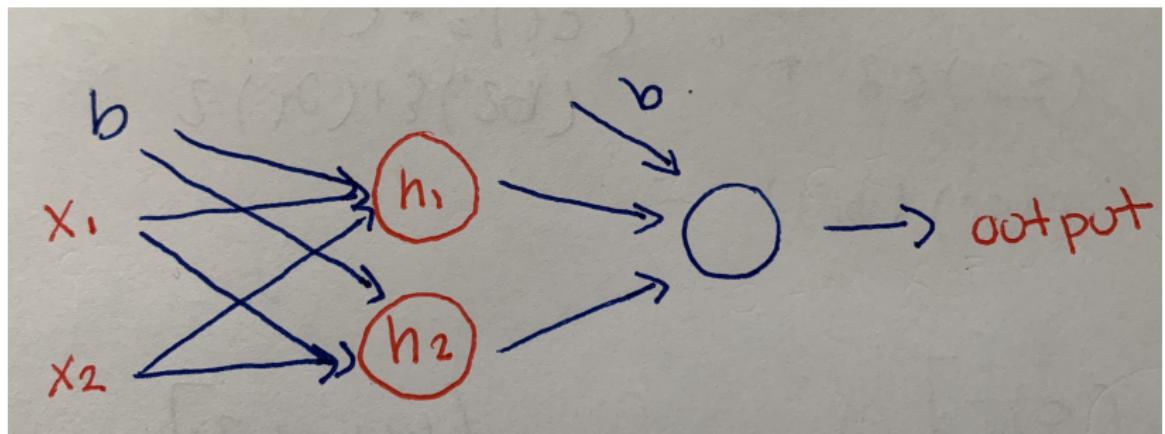
Por qué necesitamos redes de más de un nodo y más de una capa?



Solución



Pesos



Estos pesos nos dan los siguientes resultados

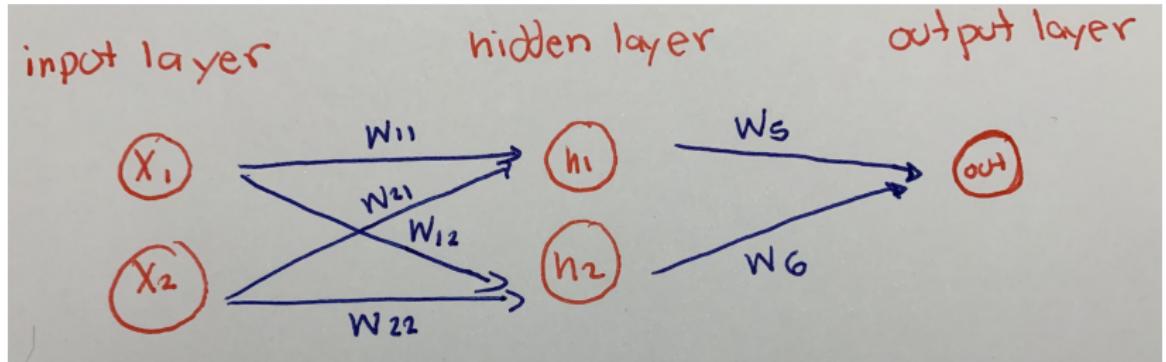
x_1	x_2	h_1	h_2	\hat{y}
0	0	0	1	0
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Intuición

- ▶ las capas intermedias calculan relaciones mas complejas
- ▶ ayudan a transformar el problema en pedazos que sean linearmente separables
- ▶ entre más profunda sea una red puede obtener(develar) relaciones mas y mas complejas

Ejemplo Backpropagation

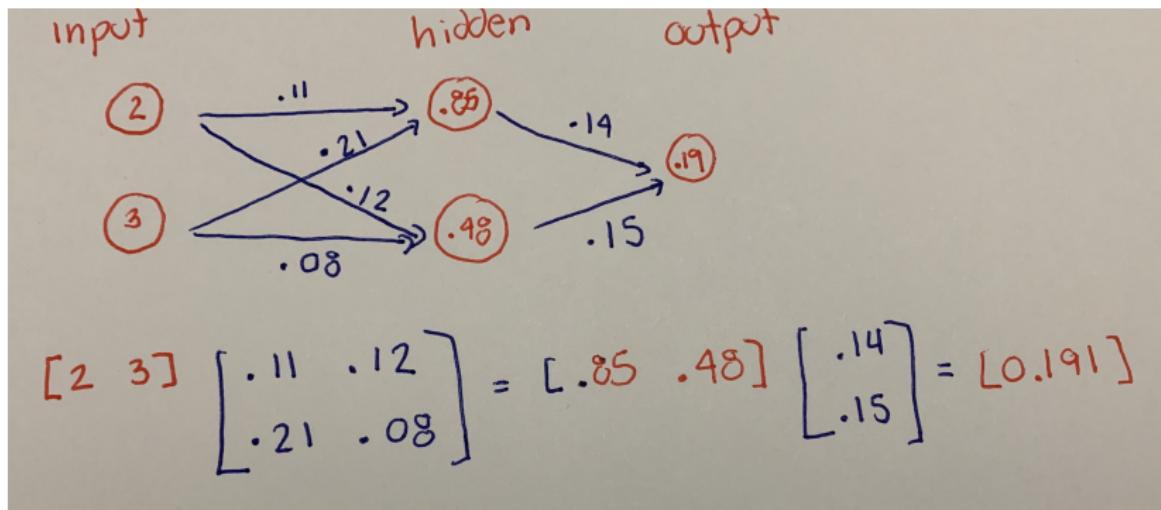
Red Neuronal



Datos

- ▶ supongamos que tenemos un solo ejemplo $x_1 = 2, x_2 = 3$ y $out = 1$
- ▶ inicializamos los pesos como
 $w_{11} = .11 \quad w_{12} = .12 \quad w_{21} = .21 \quad w_{22} = .08$ y
 $w_5 = .85 \quad w_6 = .15$
- ▶ para este ejemplo la función de activación es solo $w \cdot x$

Pase hacia adelante (feed forward)



Error de la última capa

- ▶ Asumimos función de costos cuadrática

$$C = \frac{1}{2}(predicción - real)^2 = \frac{1}{2}(0.191 - 1)^2 = 0.327$$

Cómo reducimos ese error?

- ▶ mejorando la predicción

$$\text{predicción} = \text{out}$$

$$\text{predicción} = h_1 * w_5 + h_2 * w_6$$

$$\text{predicción} = (x_1 * w_{11} + x_2 * w_{21}) * w_5 + (x_1 * w_{12} + x_2 * w_{22}) * w_6$$

Backpropagation (propagación hacia atrás)

- ▶ necesitamos cambiar los pesos de tal manera que nuestro costo se reduzca
- ▶ como hacemos esto? Gradient descent

$$w_j^* = w_j - a \frac{\partial C}{\partial w_j}$$

Backpropagation (propagación hacia atrás)

- ▶ si queremos actualizar w_5 necesitaremos $\frac{\partial C}{\partial w_5}$
- ▶ acá podemos hacerlo a mano

$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial \text{predicción}} \frac{\partial \text{predicción}}{\partial w_5}$$

- ▶ Calculamos estas dos parciales por separado

$$\frac{\partial C}{\partial \text{predicción}} = (\text{predicción} - \text{real})$$

Backpropagation (propagación hacia atrás)

$$\frac{\partial \text{predicción}}{\partial w_5} = x_1 * w_{11} + x_2 * w_{21} = h_1$$

* Entonces

$$\frac{\partial C}{\partial w_5} = (\text{predicción} - \text{actual}) * h_2 = \Delta h_2$$

Cómo actualizamos w_5

- ▶ $w_5^* = w_5 - a\Delta h_1$
- ▶ similarmente $w_6^* = w_6 - a\Delta h_2$

Y entonces como actualizamos pesos en las capas anteriores?

$$\frac{\partial C}{\partial w_{11}} = \frac{\partial C}{\partial \text{predicción}} \frac{\partial \text{predicción}}{\partial h_1} \frac{\partial h_1}{\partial w_{11}}$$

* Otra vez calculemos todas estas parciales por separado

$$\frac{\partial C}{\partial \text{predicción}} = (\text{predicción} - \text{real})$$

$$\frac{\partial \text{predicción}}{\partial h_1} = w_5$$

$$\frac{\partial h_1}{\partial w_{11}} = x_1$$

Juntamos todo

$$\frac{\partial C}{\partial w_{11}} = (\text{predicción} - \text{real})w_5x_{11} = \Delta w_5x_{11}$$

* las formulas para los otros cuatro pesos van a ser iguales

Formulas para actualizar los pesos

$$\begin{bmatrix} w_5 \\ w_c \end{bmatrix} = \begin{bmatrix} w_5 \\ w_c \end{bmatrix} - \alpha \Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$$
$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} - \alpha \Delta \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} w_5 & w_6 \end{bmatrix}$$

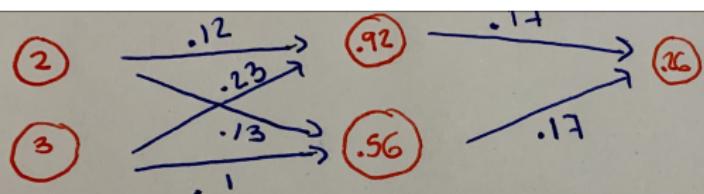
Calculamos los nuevos pesos

- ▶ asumimos que la tasa de aprendizaje es $a = 0.05$
- ▶ $\Delta = \text{predicción} - \text{real} = -0.809$

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.19 \\ 0.15 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$
$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 2 \\ 3 \end{bmatrix} \begin{bmatrix} .14 & .15 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ -.23 & .10 \end{bmatrix}$$

Mejora nuestra predicción?

► un poco



$$\begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix} = \begin{bmatrix} .92 & .56 \end{bmatrix} \begin{bmatrix} .14 \\ .17 \end{bmatrix} = [0.18]$$

Preguntas:

1. Cuáles son los pesos y el error si iteramos una vez mas?
2. Qué pasa si cambiamos la tasa de aprendizaje?
3. Qué pasa si agregamos una función de activación distinta a $f(z) = z$?

Otra iteración:

- ▶ tasa de aprendizaje $a = 0.05$
- ▶ $\Delta = \text{predicción} - \text{real} = .26 - 1 = -0.74$
- ▶ podemos usar las mismas fórmulas que antes

Otra iteración: pesos

$$\begin{bmatrix} w_5^* \\ w_6^* \end{bmatrix} = \begin{bmatrix} .17 \\ .17 \end{bmatrix} - (0.05)(-.74) \begin{bmatrix} 0.92 \\ 0.56 \end{bmatrix} = \begin{bmatrix} .20 \\ .19 \end{bmatrix}$$

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} 0.12 & 0.13 \\ 0.23 & 0.10 \end{bmatrix} - (0.05)(-.74) \begin{bmatrix} 2 \\ 3 \end{bmatrix} [0.17 \ 0.17] =$$
$$\begin{bmatrix} 0.13 & 0.14 \\ 0.24 & 0.11 \end{bmatrix}$$

Otra iteración: predicción

$$[2 \quad 3] \begin{bmatrix} .13 & .14 \\ .24 & .11 \end{bmatrix} = [.98 \quad .61] \begin{bmatrix} .20 \\ .19 \end{bmatrix} = .31$$

Cambiar la tasa de aprendizaje:

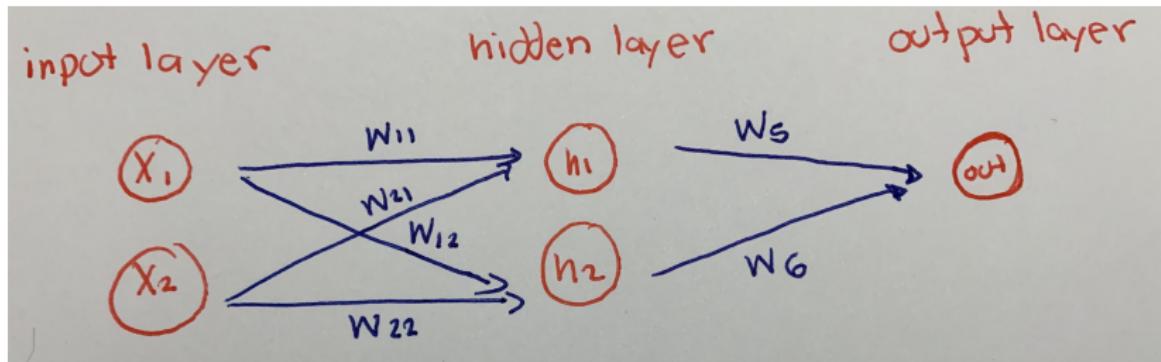
- ▶ $a = 0.25$

$$\begin{bmatrix} w_5^* \\ w_6^* \end{bmatrix} = \begin{bmatrix} .14 \\ .15 \end{bmatrix} - (.25)(-.809) \begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} .31 \\ .23 \end{bmatrix}$$

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} 0.11 & 0.12 \\ 0.21 & 0.08 \end{bmatrix} - (.25)(-.809) \begin{bmatrix} 2 \\ 3 \end{bmatrix} \begin{bmatrix} .14 & .15 \end{bmatrix} =$$
$$\begin{bmatrix} 0.16 & 0.18 \\ 0.29 & 0.17 \end{bmatrix}$$

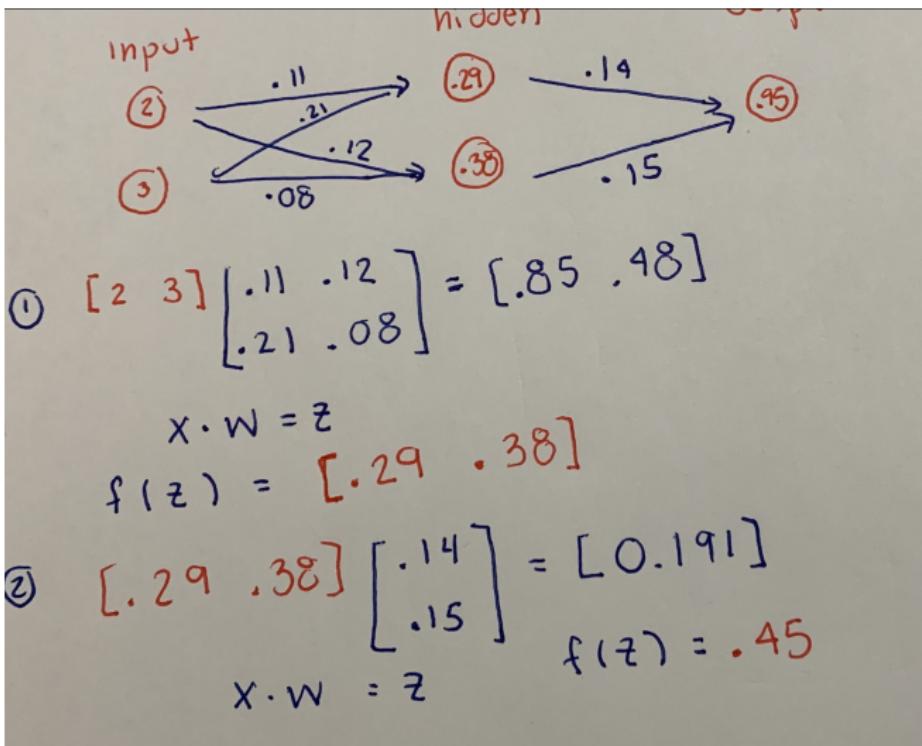
- ▶ la nueva estimación es 0.569

Cambiar la función de activación



Feed forward con Sigmoid function

$$\blacktriangleright f(z) = \frac{1}{1+e^{-z}}$$



Como hacemos la propagación hacia atras?

- ▶ Error en la última capa: $C = \frac{1}{2}(.45 - 1)^2 = 0.15125$

$$\text{predicción} = \sigma(h_1 * w_5 + h_2 * w_6)$$

$$\text{predicción} = \sigma(\sigma(x_1 w_{11} + x_2 w_{21}) * w_5 + \sigma(x_1 w_{12} + x_2 w_{22}) * w_6)$$

Backpropagation

$$\frac{\partial C}{\partial w_5} = \frac{\partial C}{\partial \text{predicción}} \frac{\partial \text{predicción}}{\partial w_5}$$

$$\frac{\partial \text{predicción}}{\partial w_5} = h_1 * \sigma'(h_1 w_5 + h_2 w_2) = \frac{h_1 e^{-(h_1 w_5 + h_2 w_2)}}{(1 + e^{-(h_1 w_5 + h_2 w_2)})^2}$$

* el concepto es el mismo solo tenemos que cargar el término $\sigma'(z)$

Cómo escribimos esto en R

Propagación hacia adelante

- ▶ combinación lineal de los inputs de entrada y los pesos
- ▶ $z_1 = X \cdot W_1 = [1 \quad x]W_1$
- ▶ aplicamos la función de activación
- ▶ $h = \sigma(z_1)$
- ▶ capa de salida : $z_2 = HW_2 = [1 \quad h]W_2$
- ▶ función de activación $\hat{y} = \sigma(z_2)$
- ▶ entonces

$$\hat{y} = \sigma(HW_2) = \sigma([1 \quad \sigma(XW_1)]W_2)$$

Propagación hacia adelante

```
sigmoid <- function(x){  
  return(1/(1+exp(-x)))  
}  
  
feedforward <- function(x,w1,w2){  
  z1 <- cbind(1,x) %*% w1  
  h <- sigmoid(z1)  
  z2 <- cbind(1,h) %*% w2  
  list(output = sigmoid(z2), h = h)  
}
```

Propagación hacia atras

$$C = \frac{1}{2n} \sum_n (y - \hat{y})^2$$

Gradient descent implica que

$$W^* = W - a \nabla C(W)$$

$$\frac{\partial C}{\partial W_2} = \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_2}$$

Donde:

$$\frac{\partial C}{\partial \hat{y}} = (y - \hat{y})$$

$$\frac{\partial \hat{y}}{\partial W_2} = H^T \sigma(HW_2)(1 - \sigma(HW_2)) = H^T \hat{y}(1 - \hat{y})$$

Nota:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Propagación hacia atras

$$\frac{\partial C}{\partial W_1} = \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial H} \frac{\partial H}{\partial W_1}$$

$$\frac{\partial \hat{y}}{\partial H} = \sigma'(HW_2)W_2^T = \hat{y}(1 - \hat{y})W_2^T$$

$$\frac{\partial H}{\partial W_1} = X^T [0 \quad \sigma(XW_1)(1 - \sigma(XW_1))]$$

Propagación hacia atras

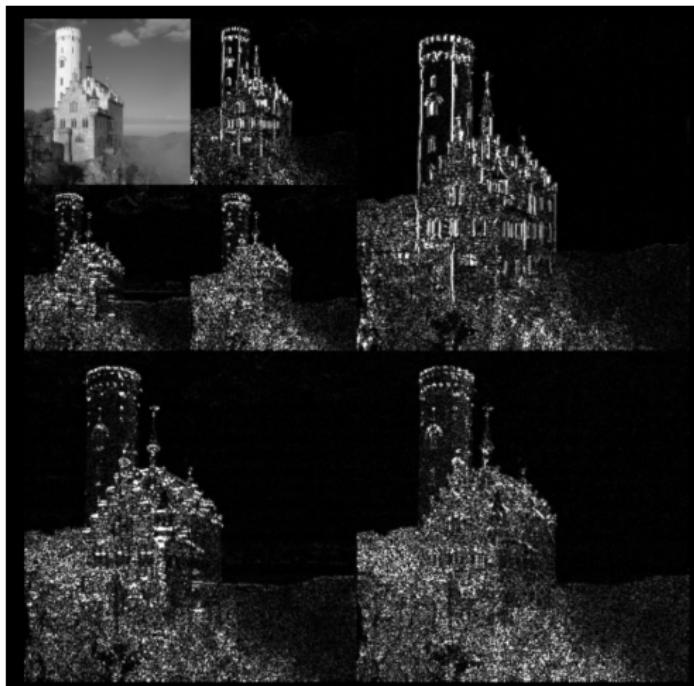
```
backpropagate <- function(x,y,y_hat,w1,w2,h,a){  
  dw2 <- 2*t(cbind(1,h)) %*% (y_hat*(1 - y_hat)*(y_hat - y))  
  dh <- (y_hat-y) %*% t(w2[-1, drop = FALSE])  
  dw1 <- t(cbind(1,x)) %*% (h*(1-h)*dh)  
  w1 <- w1 - a*dw1  
  w2 <- w2 - a*dw2  
  list(w1 = w1 , w2 = w2)  
}
```

Aprendizaje

```
learn <- function(x, y , hidden = 5, a = 0.01, iterations = 1000) {
  d <- ncol(x) + 1
  w1 <- matrix(rnorm(d*hidden),d,hidden)
  w2 <- as.matrix(rnorm(hidden+1))
  for (i in 1:iterations){
    ff <- feedforward(x,w1,w2)
    bp <- backpropagate(x,y,
                         y_hat = ff$output,
                         w1,w2,
                         h = ff$h, a = a)
    w1 <- bp$w1
    w2 <- bp$w2
  }
  list(output = ff$output , w1 = w1, w2 = w2)
}
```

Vamos a probar nuestra red

- ▶ Datos : fotos de billetes clasificado como falsos o no dependiendo de la varianza, skewness, curtosis y entropía de la transformación de Wavelnet de la imagen.



Vamos a probar nuestra red

- ▶ Datos: estan en el repo de la clase

```
bank <- read_csv('banknotes.txt')  
head(bank)
```

```
## # A tibble: 6 x 5  
##   variance skewness curtosis entropy classification  
##       <dbl>     <dbl>     <dbl>     <dbl>          <dbl>  
## 1     3.62      8.67    -2.81    -0.447          0  
## 2     4.55      8.17    -2.46    -1.46          0  
## 3     3.87     -2.64     1.92     0.106          0  
## 4     3.46      9.52    -4.01    -3.59          0  
## 5     0.329     -4.46     4.57    -0.989          0  
## 6     4.37      9.67    -3.96    -3.16          0
```

Probar nuestra red

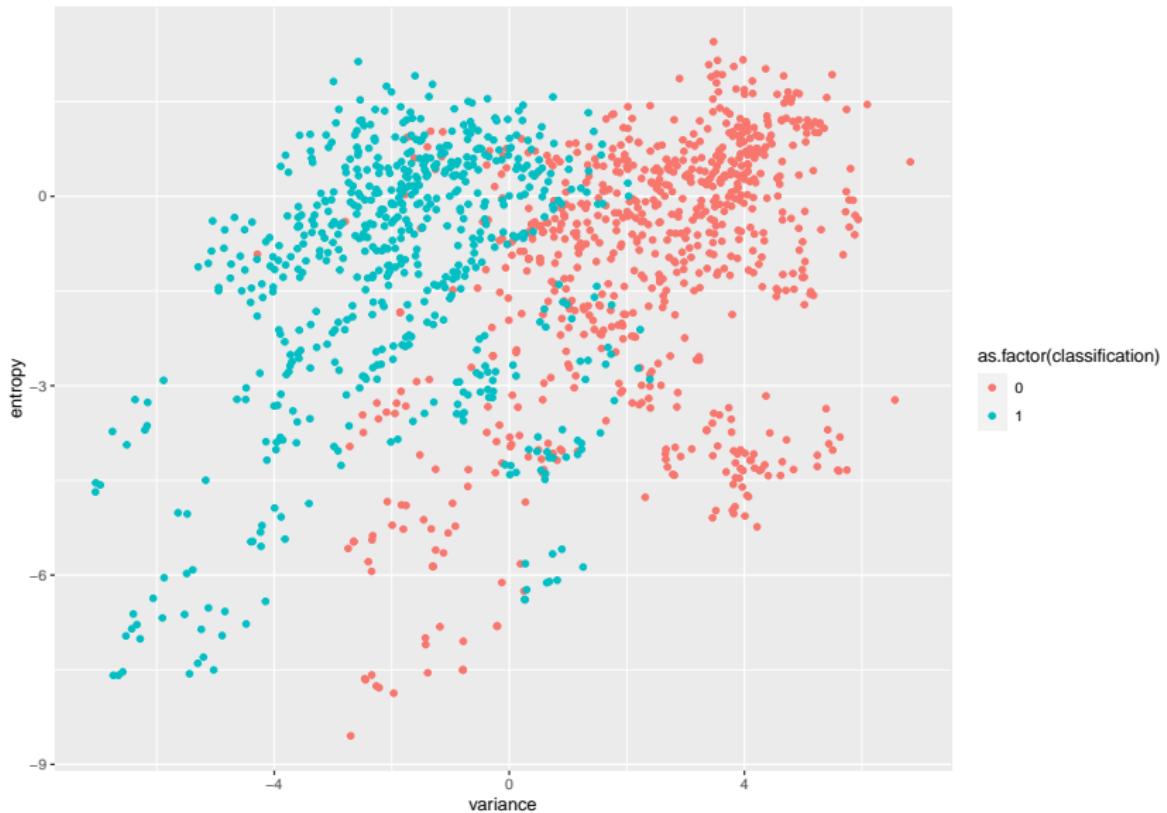
```
x <- data.matrix(bank[,c("variance","entropy")])
y <- as.numeric(bank$classification)
nnet5 <- learn(x,y, hidden = 5, iterations = 10000)
```

- ▶ cómo calculamos que tan bien lo hizo?

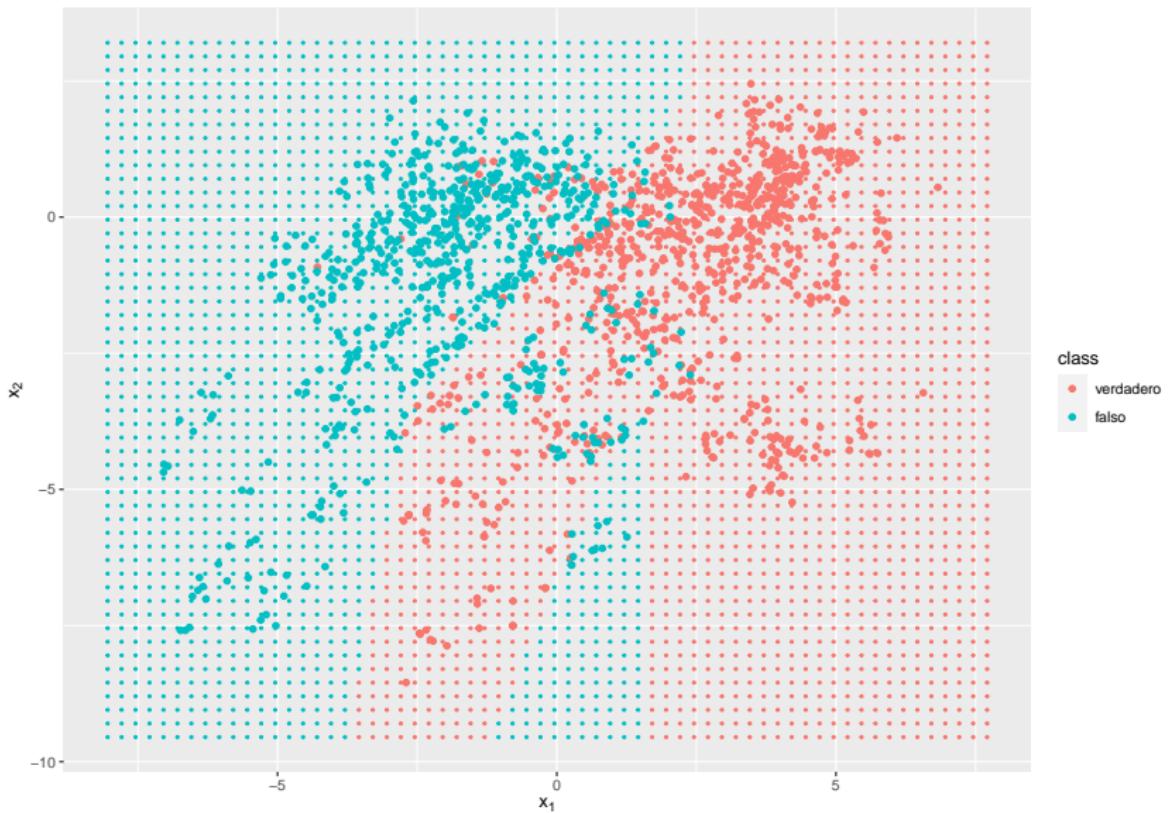
```
mean((nnet5$output > .5) == y)
```

```
## [1] 0.8979592
```

Cómo se ven nuestros datos graficamente



Dibujando las fronteras de decisión



Qué le falta a nuestro algoritmo?

- ▶ nunca incluimos explícitamente la función de costos
- ▶ funciona bien en nuestro ejemplo:
 - ▶ que pasa si intentamos usarlo con una base de datos más grande
 - ▶ con una clasificación que no sea binaria
- ▶ podemos mejorarlo pero si ya está claro podemos usar una librería que nos da más flexibilidad que nuestro código escrito a mano

Keras y Tensorflow

```
install.packages("tensorflow")
library(tensorflow)
install_tensorflow()
install.packages("keras")
library(keras)
install_keras()
```

Definimos el modelo usando keras

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 5, activation = "sigmoid", input_shape = c(4))
  layer_dense(units = 1, activation = "sigmoid")
summary(model)
```

```
## Model: "sequential"
## -----
## Layer (type)          Output Shape
## =====
## dense_1 (Dense)      (None, 5)
## -----
## dense (Dense)         (None, 1)
## =====
## Total params: 21
## Trainable params: 21
## Non-trainable params: 0
## -----
```

Declaramos: función de costos, tasa de aprendizaje y métrica

- ▶ ojo nuestro resultado va a ser distinto al de Keras, por qué?

```
model %>% compile(  
  loss = 'mean_squared_error',  
  optimizer = "sgd",  
  metrics = c('accuracy'))  
)
```

Entrenamos el modelo

```
fit_bank <- model %>% fit(  
  x, y,  
  epochs = 1000,  
  batch_size = 10,  
  validation_split = 0.2,  
  verbose=0  
)
```

Gráfica

```
plot(fit_bank)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

