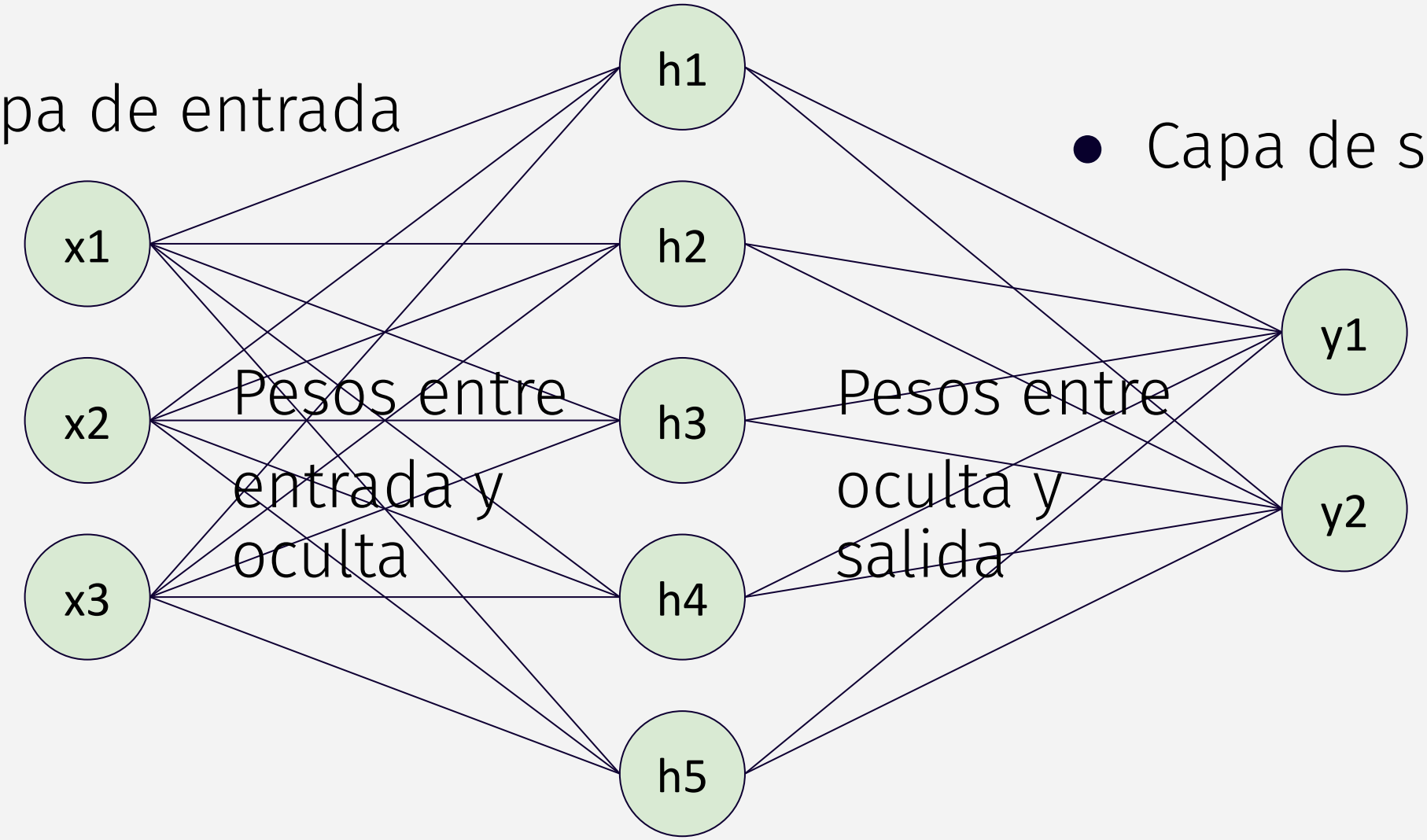
Aprendizaje Automático Profundo (Deep Learning)

Dr. Facundo Quiroga - Dr. Franco Ronchetti

Capa Dropout

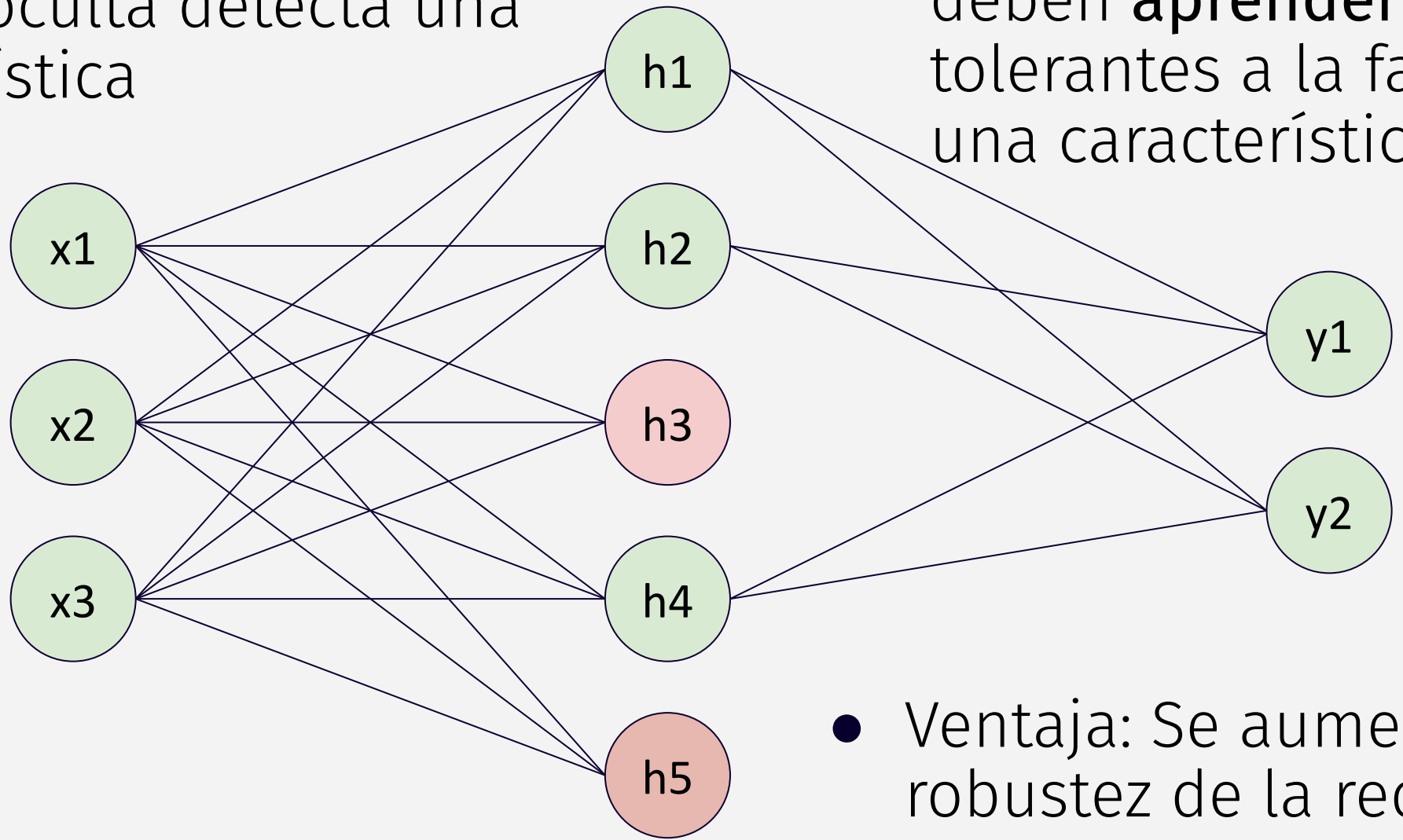
Red de 2 capas lineales vista “por neuronas”

- Capa de entrada
- Capa *oculta*
- Capa de salida



Dropout: desactivar neuronas de forma aleatoria

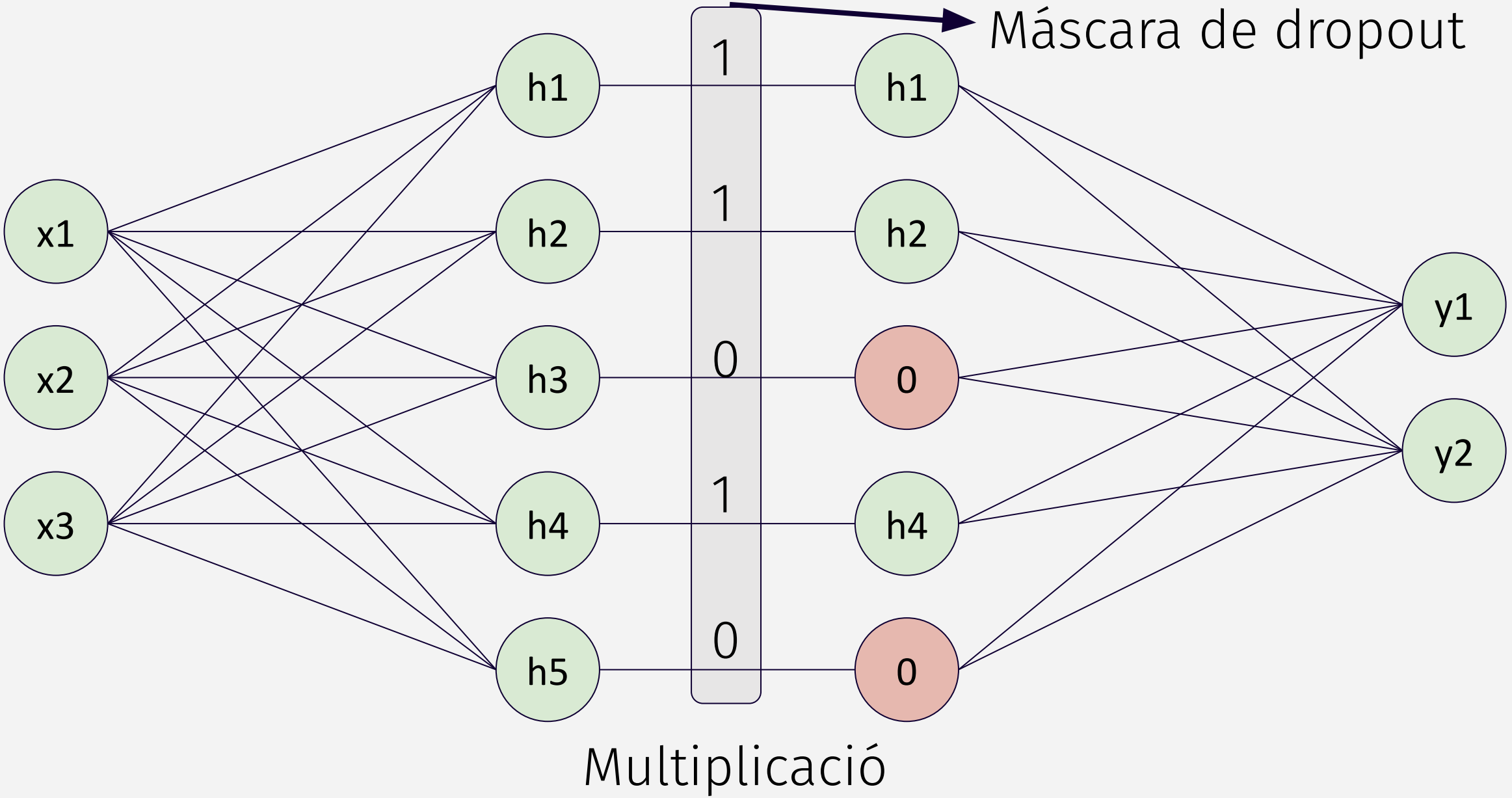
- Asunción: cada neurona de la capa oculta detecta una característica



- Neuronas de salida deben **aprender** a ser tolerantes a la falta de una característica

- Ventaja: Se aumenta la robustez de la red

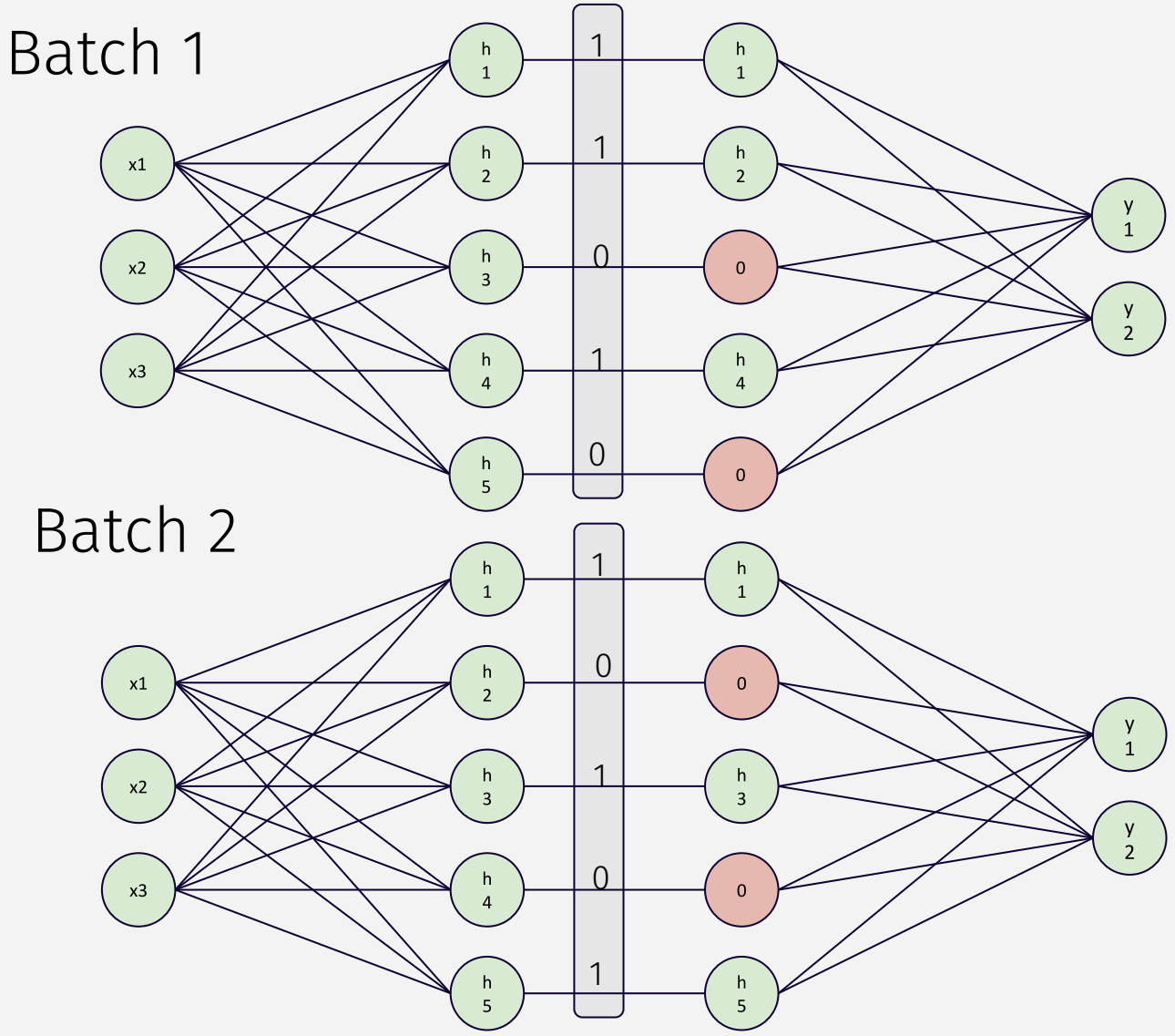
Dropout: Implementación



Dropout y batchs

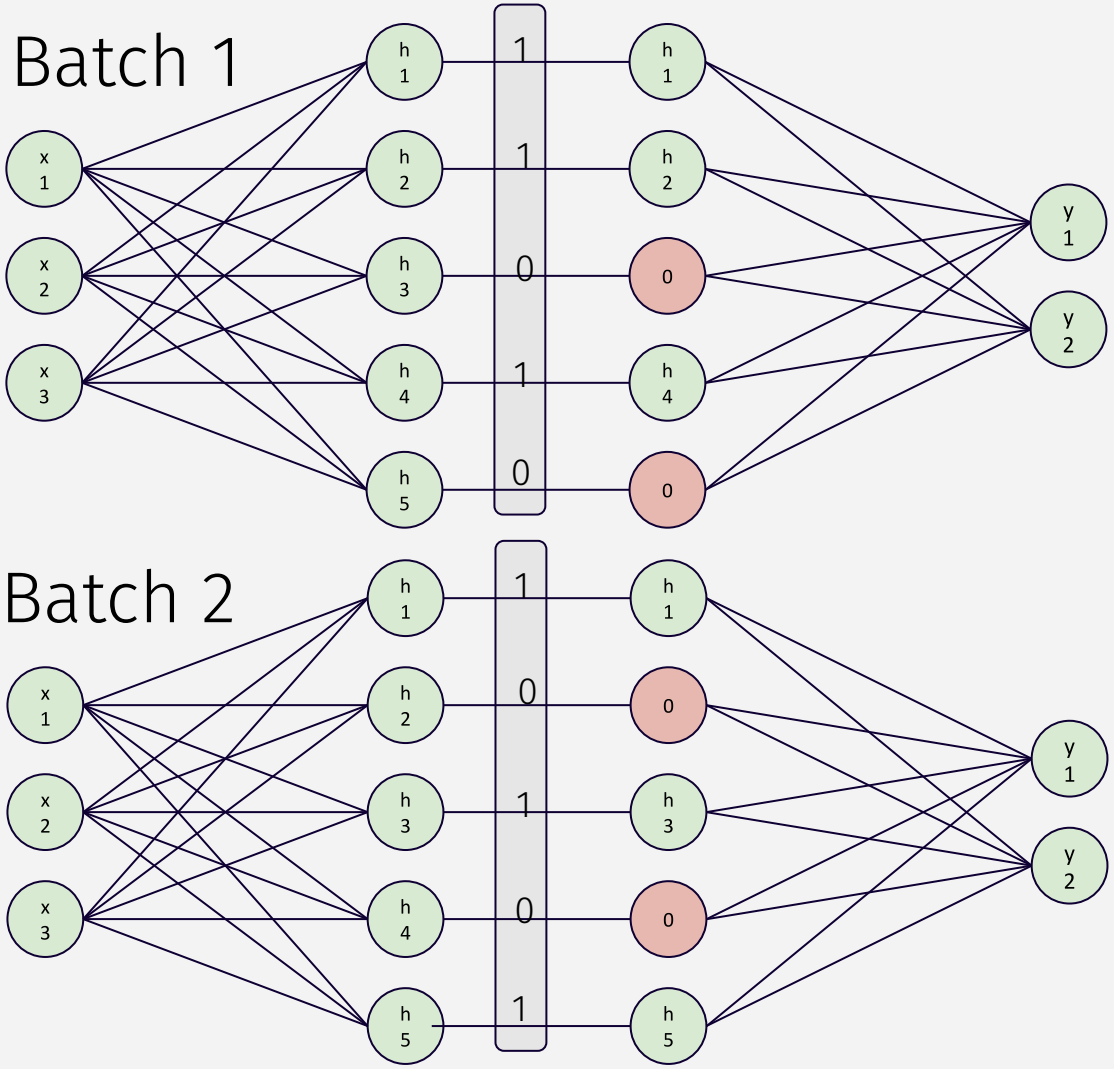
- Las neuronas a activar **cambian** en **cada batch**
  - La máscara de Dropout se recalcula
    - Cada neurona se activa con una probabilidad ***p***
    - 0<**p**<1 es hiperparámetro

```
def dropout_mask(n,p):
    m = np.zeros(n)
    for i in range(n):
        if random.rand()<p:
            m[i]=1
    return m
```

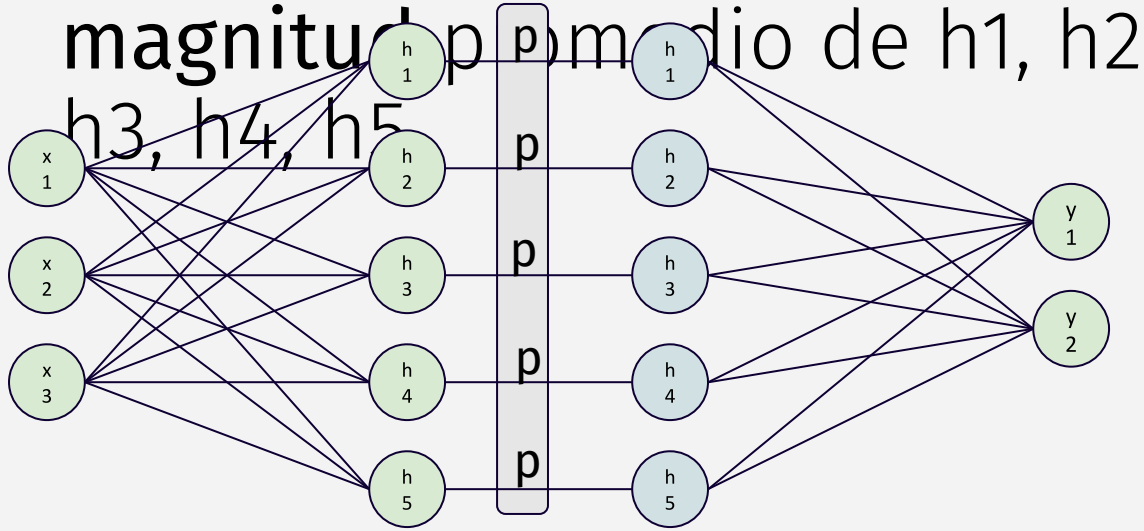


Dropout: Problema de normalización

- Durante el entrenamiento:



- Al entrenar
  - **y1** e **y2** están *acostumbradas* a tener **5*p<=5** neuronas *activadas*
- Al predecir:
  - Las activaciones se multiplican por **p**
  - Mantiene **constante** la **magnitud** promedio de h1, h2, h3, h4, h5



Implementación por fases

- Comportamiento distinto en train /test
- Poco coste computacional
- Regulariza el modelo:
  - Activaciones aleatorias
- **Modelo distinto en cada batch!**
  - Anular ciertas salidas efectivamente cambia la arquitectura.
- Valores de *p*: más bajo en capas más profundas
  - Típicamente:0.9, 0.7, 0.5

```
def dropout(act,p,phase):
   # act: activation vector
   # p: probability to keep act
   if phase == "test":
      return act*p
   elif phase == "train":
      n = act.shape[0]
      mask = dropout_mask(n,p)
      return act*mask
```

Dropout en Keras (notebook)

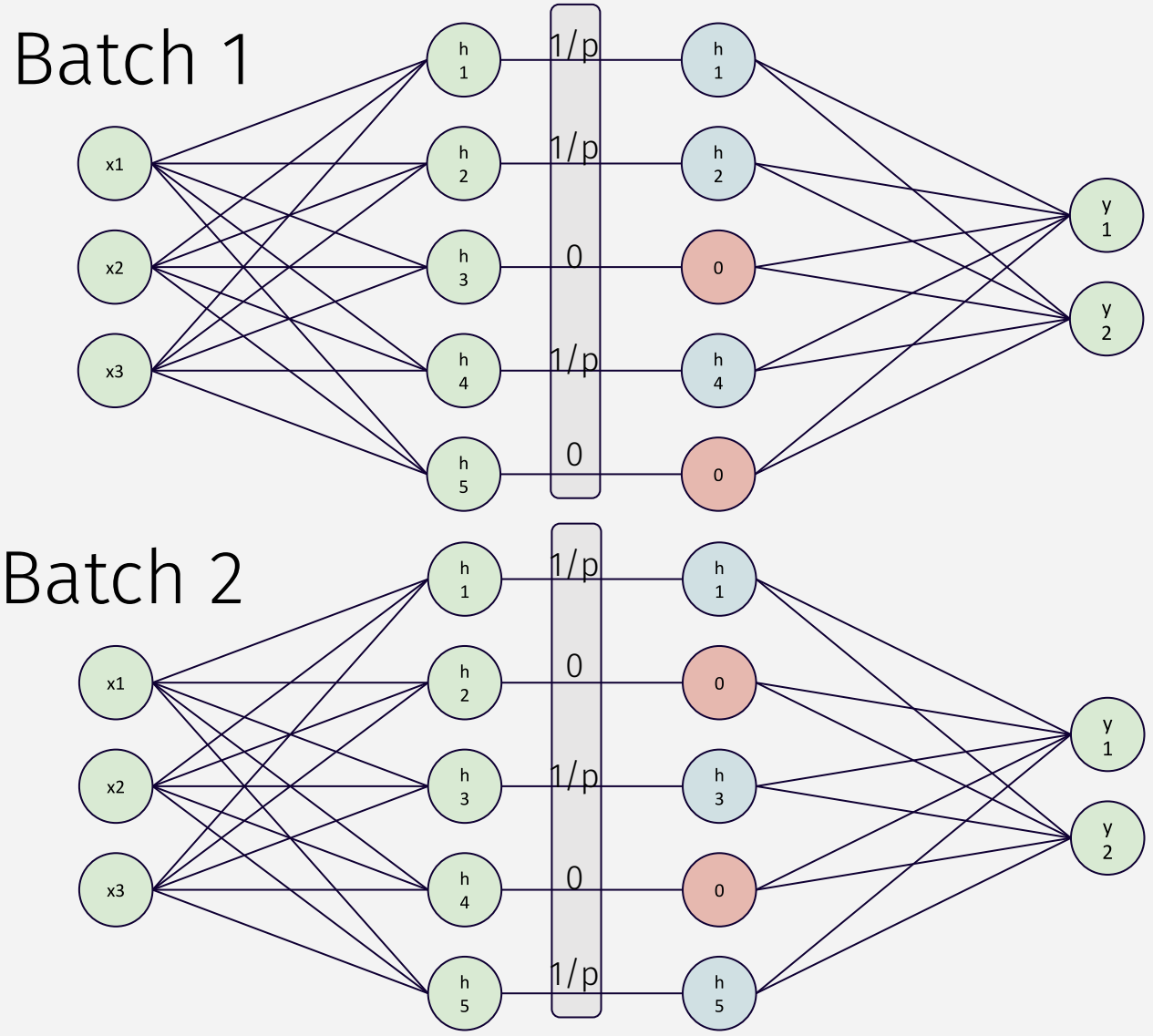
```
input_shape=(32,32,3)
classes=10


model = Sequential()
model.add(Conv2D(32,(3,3),input_shape=input_shape,..))
model.add(Flatten())
model.add(Dense(64,activation="relu"))
# Agrego Dropout con probabilidad de 0.7
model.add(Dropout(1-0.7)) #recibe 1-p
model.add(Dense(128,activation="relu"))
# Agrego Dropout con probabilidad de 0.5
model.add(Dropout(1-0.5)) #recibe 1-p
model.add(Dense(classes,activation="softmax"))
print(model.summary())
```

Dropout - Alternativa

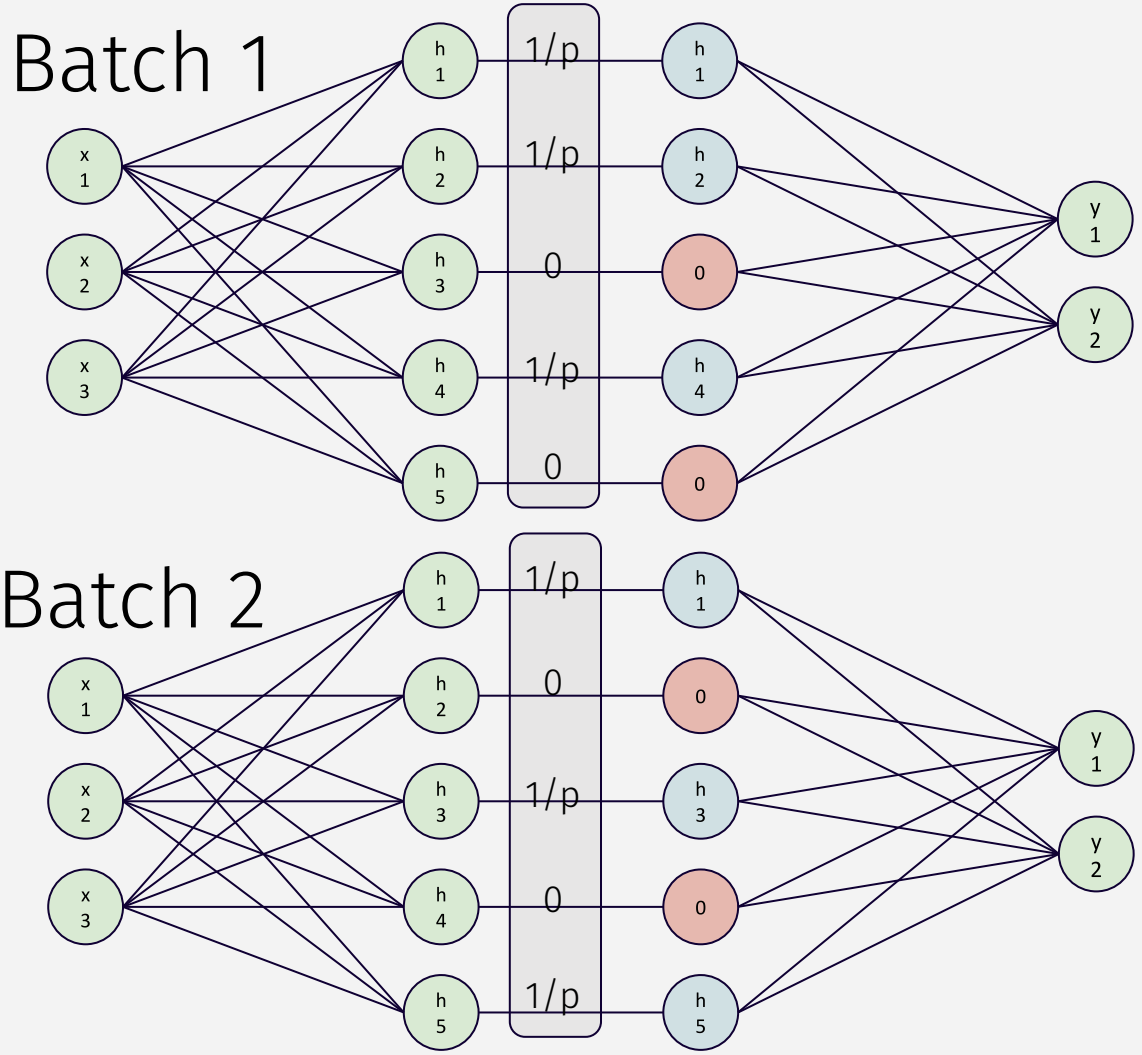
- Implementacion alternativa:
  - En la predicción, **NO** multiplicar activaciones por *p*
  - En entrenamiento, multiplicar activaciones activadas por *1/p*

```
def dropout_mask2(n,p):
    m = np.zeros(n)
    for i in range(n):
        if random.rand()<p:
            m[i]=1/p
    return m
```

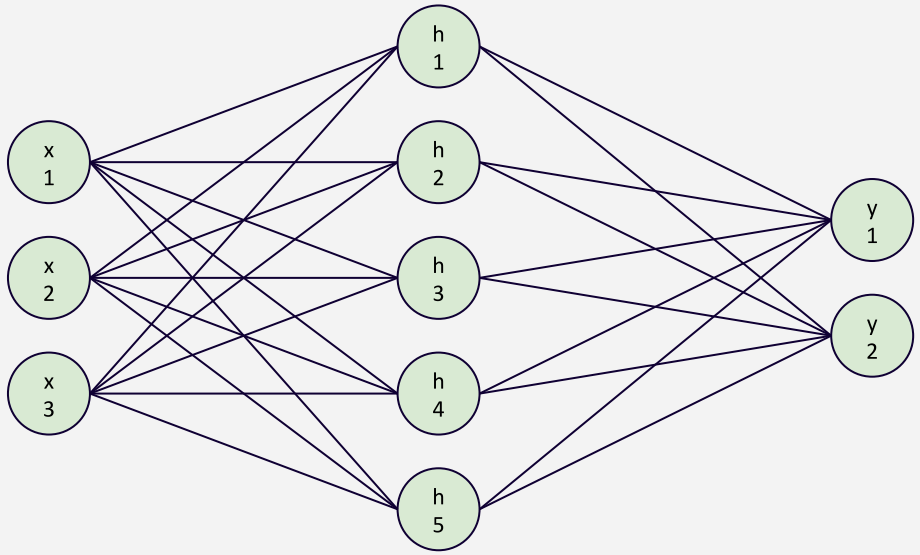


Dropout - Alternativa

- Durante el entrenamiento:



- Al entrenar
  - **y1** e **y2** están *acostumbradas* a tener la magnitud de **5*p*1/p = 5 neuronas**
- Al predecir:
  - No hacemos nada!
  - Podemos “quitar” la capa



Dropout - Alternativa

```
def dropout2(act,p,phase):

   # act: activation vector
   # p: probability to keep act

   if phase == "test":
      # no hacer nada
      return act
   elif phase == "train":
      n = act.shape[0]
      mask = dropout_mask2(n,p)
      return act*mask
```