diplomatura universitaria en
inteligencia artificial

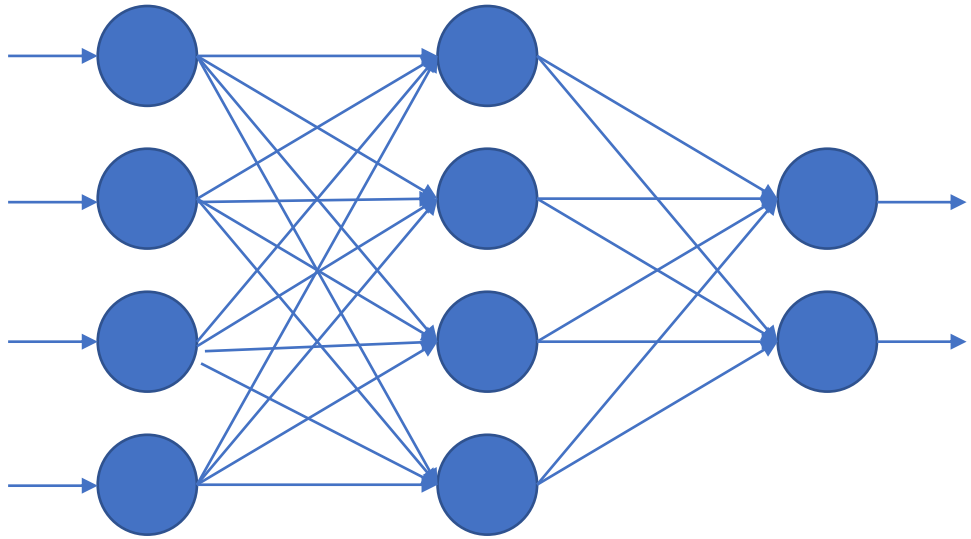
FACULTAD DE CIENCIAS
EXACTAS
UNIVERSIDAD NACIONAL DEL CENTRO
DE LA PROVINCIA DE BUENOS AIRES

Redes Neuronales

Perceptrón multicapas, Regularización, Dropout, Bias y Varianza

Agenda

- Introducción a Keras
- El problema del XOr
- Perceptrón multicapaz y el aproximado universal
- Relación entre los errores en entrenamiento y test
  - Bias / Underfitting
  - Varianza / Overfitting
- Mitigación del Overfitting
  - Regularización
  - Dropout
  - Noise.



Keras

Una API diseñada apara humanos

Keras es una API construida sobre TensorFlow para diseñar redes neuronales

- Enfocada en la simplicidad del código
- Permite acceso a recurso como GPU y TPU
- Permite trabajar con cualquier función diferenciable
- Fácilmente escalable
- Se pueden exportar lo resultados a diversos ambientes (servidores, navegadores, y móviles)

Keras

El ejemplo de MNIST

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

i = Input((size,))
d = Dense(10, activation='softmax')(i)

model = Model(inputs=i, outputs=d)
model.compile(loss='categorical_crossentropy', optimizer='sgd’)

model.summary()

model.fit(x_train, yc_train, batch_size=50, epochs=100)
```

Keras

El ejemplo de MNIST

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
i = Input((size,))
d = Dense(10, activation='softmax')(i)

model = Model(inputs=i, outputs=d)
model.compile(loss='categorical_crossentropy',
              optimizer=SGD(learning_rate=0.01, momentum=0.9))
model.summary()

model.fit(x_train, yc_train, batch_size=50, epochs=100)
```

Sumario del Modelo

Capa	Tamaño Salida	# Parámetros
input_1 (InputLayer)	[(None, 784)]	0
dense (Dense)	(None, 10)	7850

Sumario del Modelo

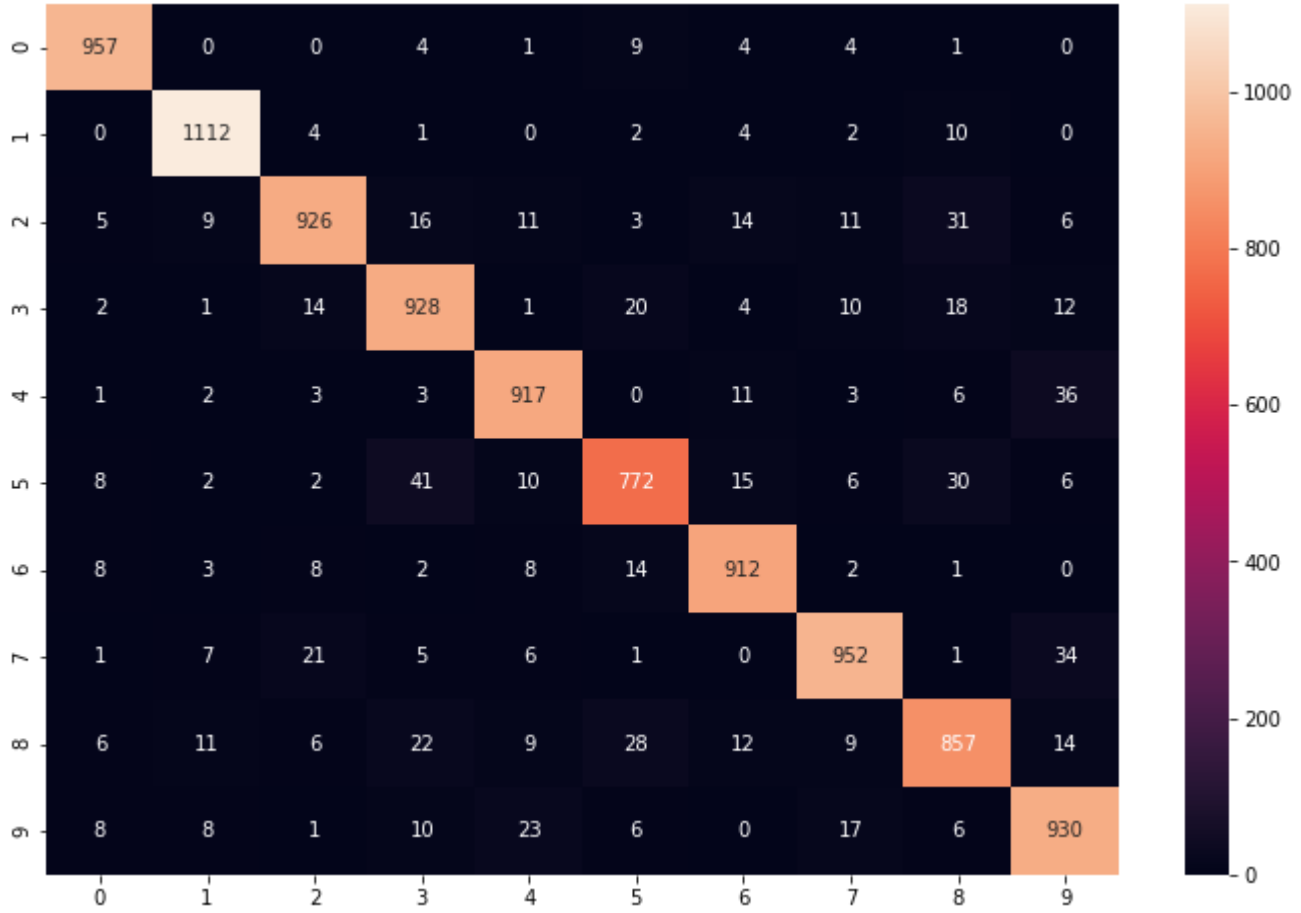
Capa	Tamaño Salida	# Parámetros
input_1 (InputLayer)	[(None, 784)]	0
dense (Dense)	(None, 10)	7850

None: No se conoce a priori (cantidad de instancias)

W: 784x10 = 7840
B: 10 = 10

Keras

El ejemplo de MNIST



El problema del XOr

Dataset Xor y regresión logística

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$$0w_1 + 0w_2 + b = 0$$
$$0w_1 + 1w_2 + b = 1$$
$$1w_1 + 0w_2 + b = 1$$
$$1w_1 + 1w_2 + b = 0$$

El problema del XOr

Dataset Xor y regresión logística

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$$0w_1 + 0w_2 + b = 0 \Rightarrow b = 0$$

$$1w_1 + 1w_2 + b = 0 \Rightarrow w_1 + w_2 = 0 \Rightarrow w_1 = -w_2$$

$$0w_1 + 1w_2 + b = 1 \Rightarrow w_2 = 1$$

$$1w_1 + 0w_2 + b = 1 \Rightarrow w_1 = 1$$

El problema del XOr

Dataset Xor y regresión logística

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$$0w_1 + 0w_2 + b = 0 \Rightarrow b = 0$$

$$1w_1 + 1w_2 + b = 0 \Rightarrow w_1 + w_2 = 0 \Rightarrow w_1 = -w_2$$

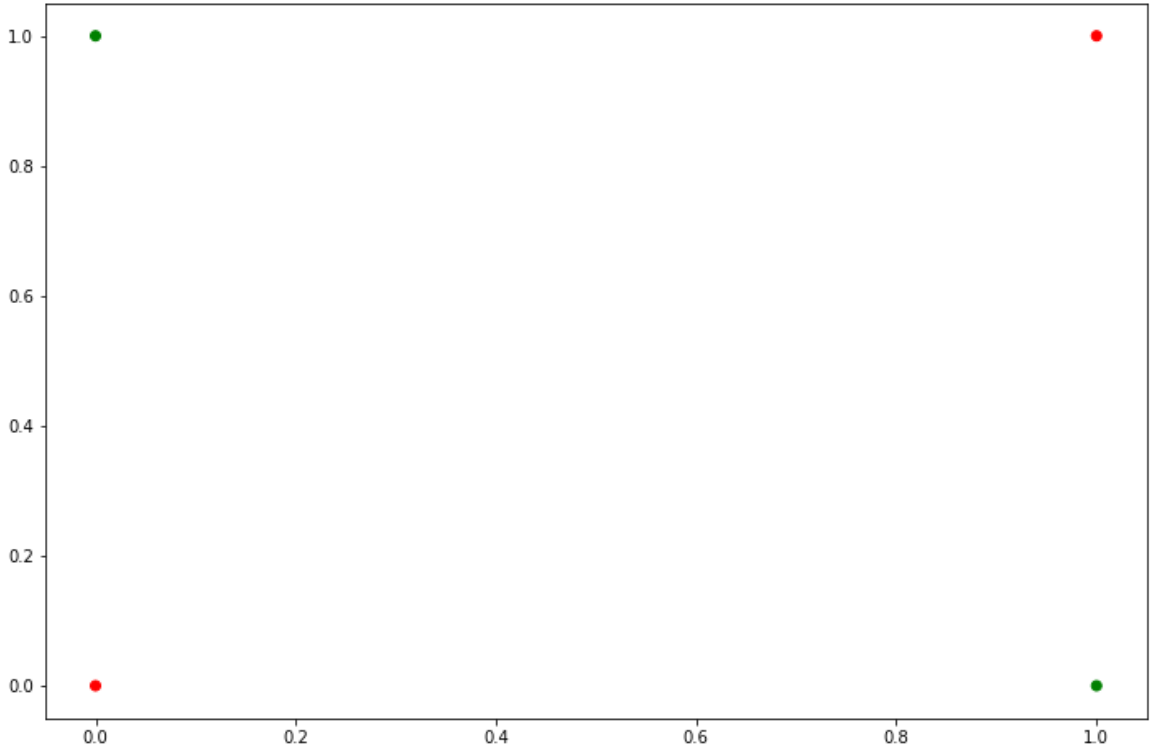
$$0w_1 + 1w_2 + b = 1 \Rightarrow w_2 = 1$$

$$1w_1 + 0w_2 + b = 1 \Rightarrow w_1 = 1$$

El problema del XOr

Dataset Xor y regresión logística

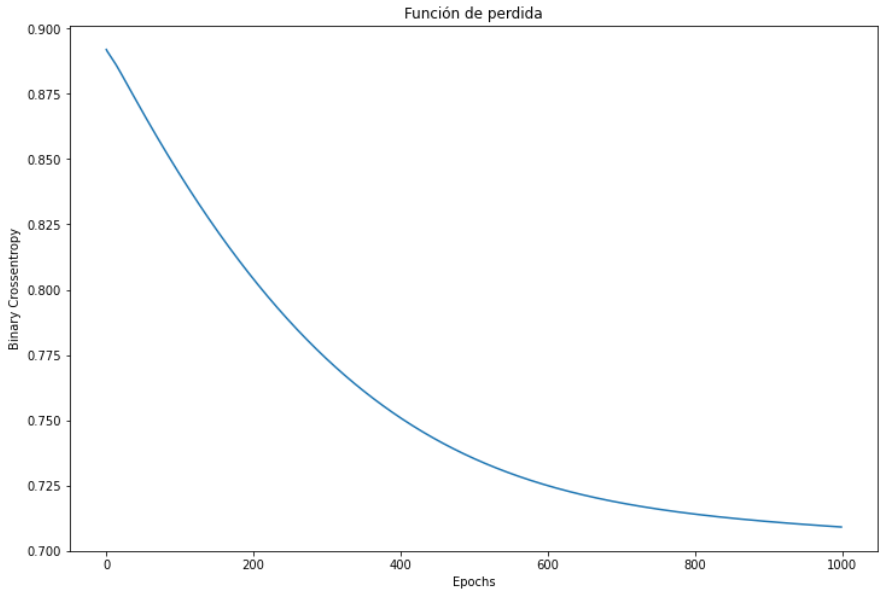
X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

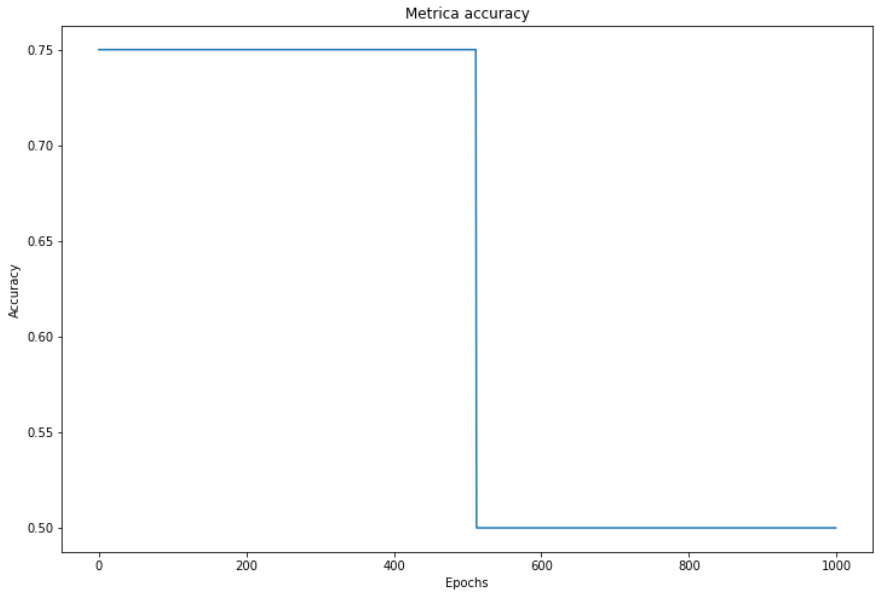


El problema del XOr

Dataset Xor y regresión logística

```
i = Input((2,))
d = Dense(1, activation='sigmoid')(i)
model = Model(i, d)
model.compile(loss='binary_crossentropy', optimizer='nadam', metrics=['accuracy'])

h = model.fit(x, y, epochs=1000, verbose=0)
```



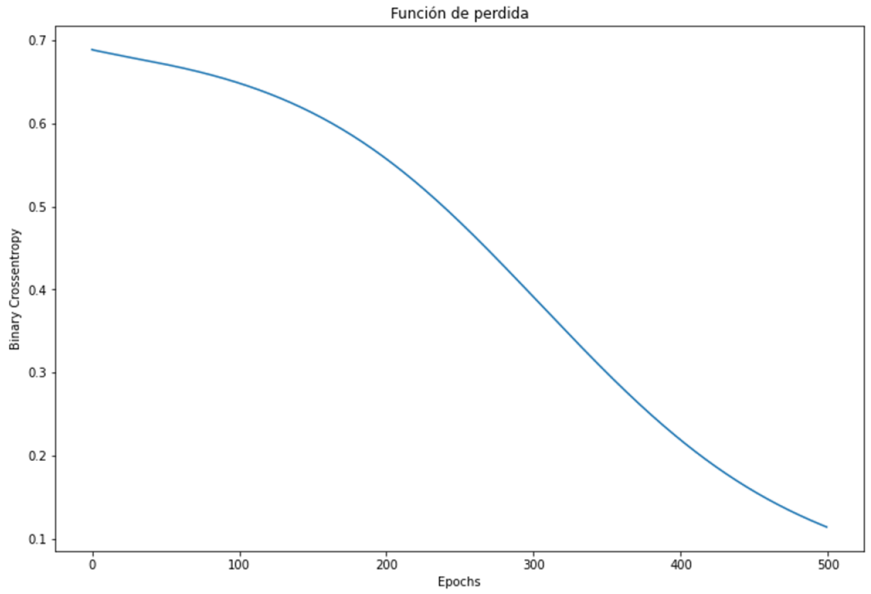


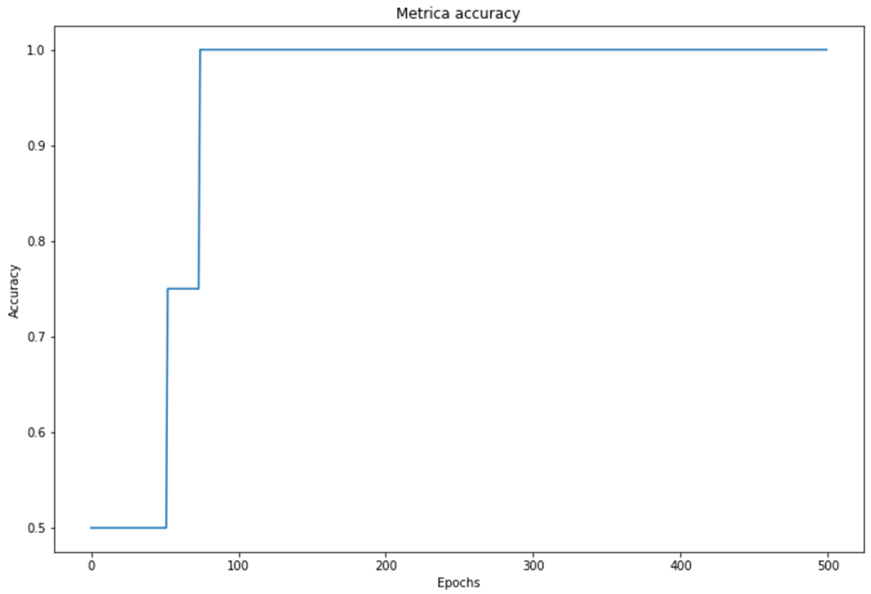
X1	X2	Yp
0	0	0.5
0	1	0.5
1	0	0.5
1	1	0.5

El problema del XOr

Dataset Xor y regresión logística

```
i = Input((2,))
d = Dense(30, activation='tanh')(i)
d = Dense(1, activation='sigmoid')(d)
model = Model(i, d)
model.compile(loss='binary_crossentropy', optimizer='nadam', metrics=['accuracy'])

h = model.fit(x, y, epochs=500, verbose=0)
```

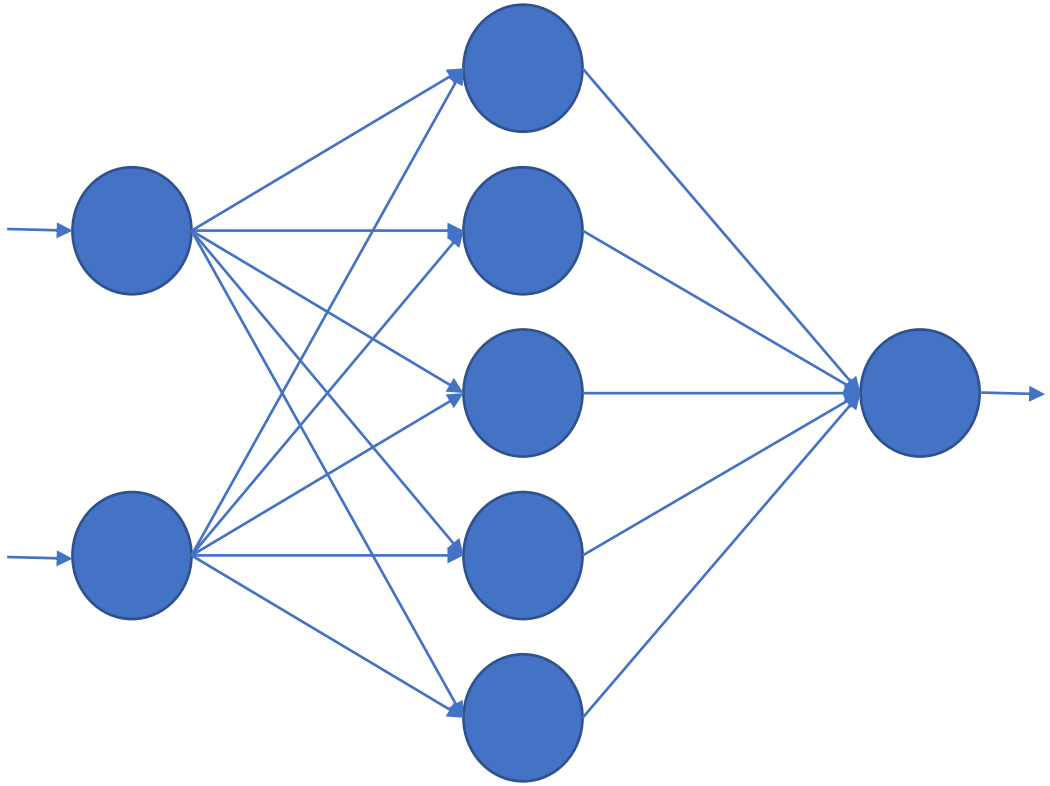




X1	X2	Yp
0	0	0.1
0	1	0.9
1	0	0.9
1	1	0.1

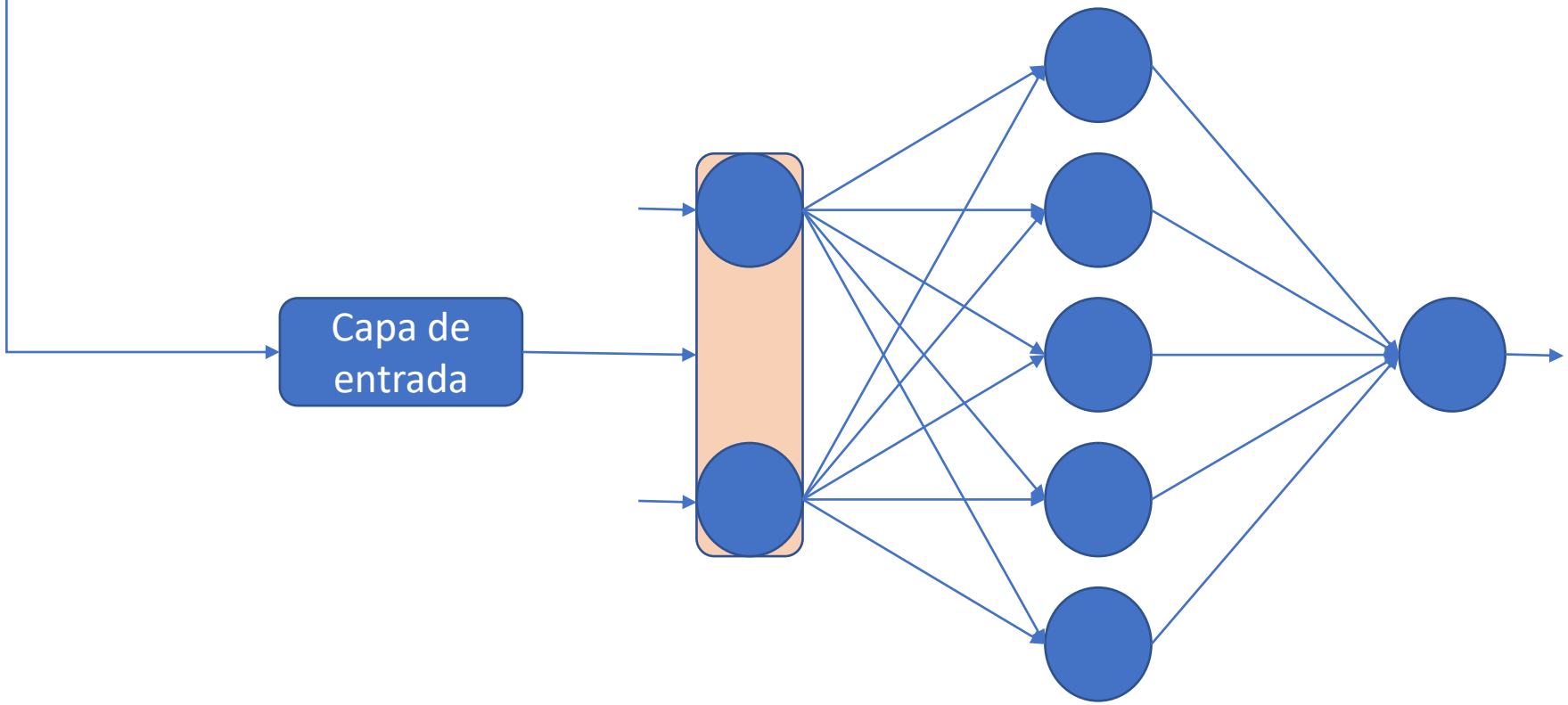
Le perceptrón multi capas

```
i = Input((2,))
d = Dense(30, activation='tanh')(i)
d = Dense(1, activation='sigmoid')(d)
model = Model(i, d)
```



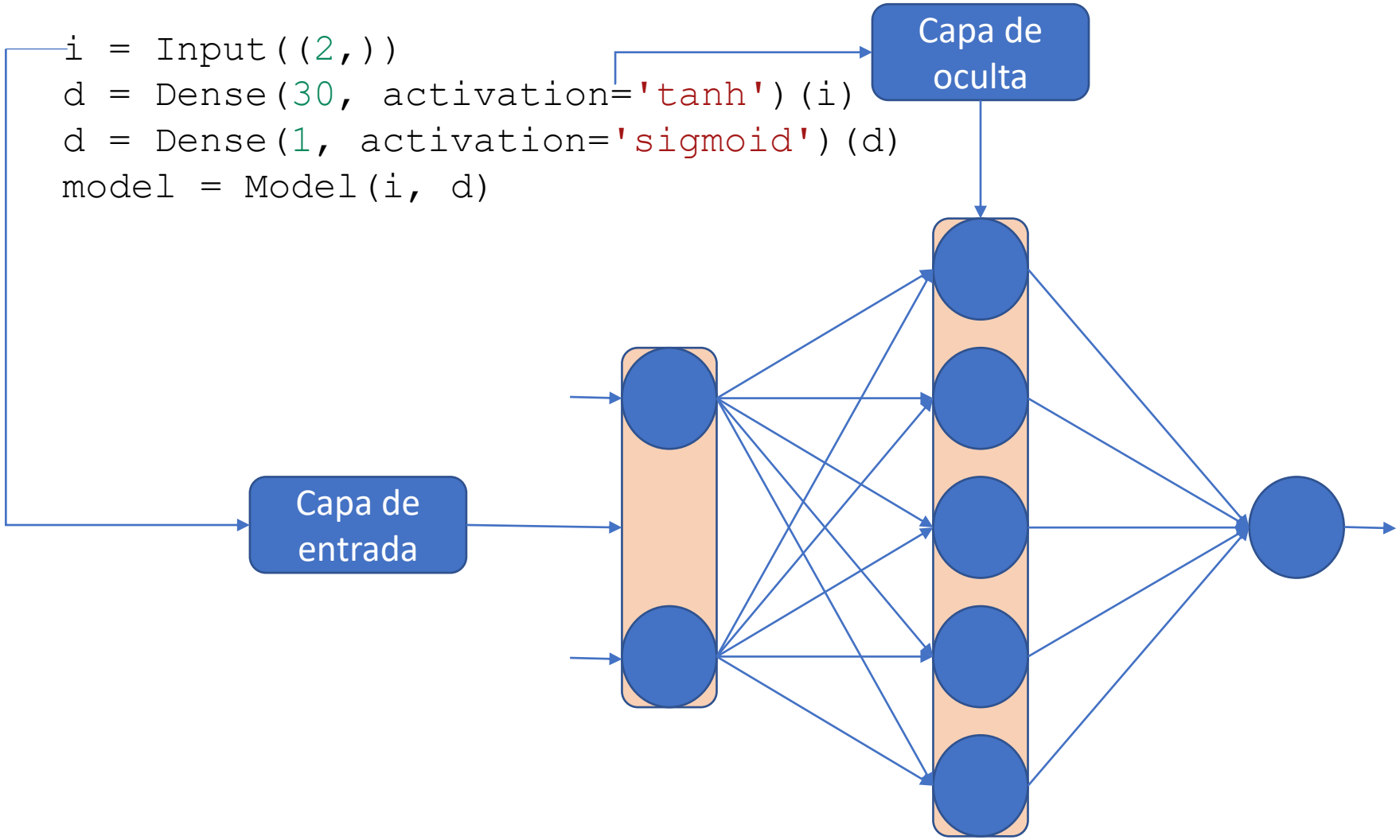
Le perceptrón multi capas

```
i = Input((2,))
d = Dense(30, activation='tanh')(i)
d = Dense(1, activation='sigmoid')(d)
model = Model(i, d)
```



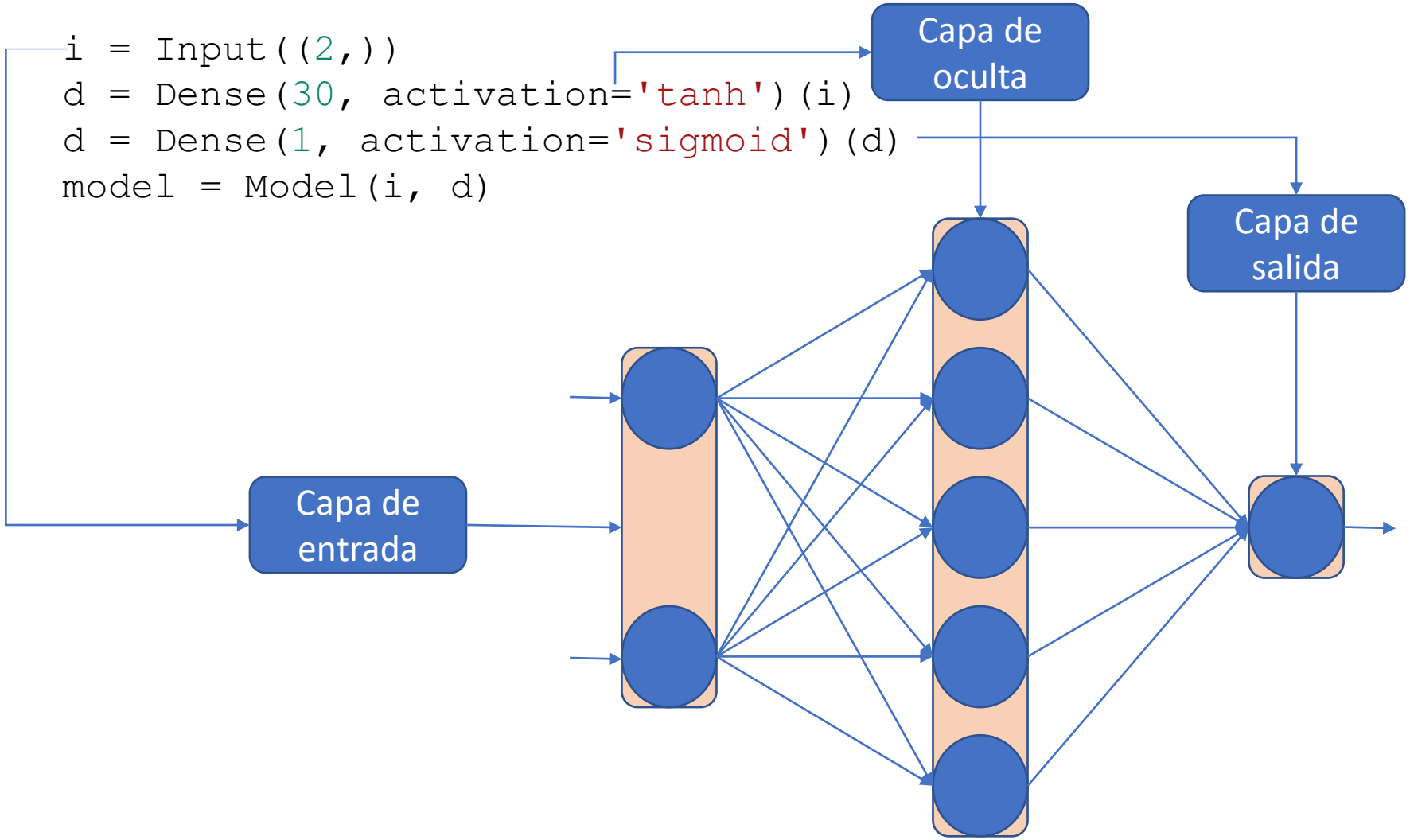
Le perceptrón multi capas

```
i = Input((2,))
d = Dense(30, activation='tanh')(i)
d = Dense(1, activation='sigmoid')(d)
model = Model(i, d)
```



Le perceptrón multi capas

```
i = Input((2,))
d = Dense(30, activation='tanh')(i)
d = Dense(1, activation='sigmoid')(d)
model = Model(i, d)
```



Universal Aproximation Theorem

Redes multi capas como aproximadores

Dado:

- $F(x) = \sum_{i=1}^{N} v_i \varphi(wx + b)$
- $f(x)$ una función arbitraria en $f: \mathbb{R}^n$

Dado un ϵ arbitrario, se pueden seleccionar un N y sus correspondientes $v_1, \ldots, v_i$ y w tal que:

- $\int |F(x) - f(x)|\, dx < \epsilon$

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control Signal Systems* **2,** 303–314 (1989). https://doi.org/10.1007/BF02551274

Universal Aproximation Theorem

Redes multi capas como aproximadores

Dado:

- $F(x) = \sum_{i=1}^{N} v_i \varphi(wx + b)$
- $f(x)$ una función arbitraria en $f: \mathbb{R}^n$

Dado un ϵ arbitrario, se pueden seleccionar un N y sus correspondientes $v_1, \ldots, v_i$ y w tal que:

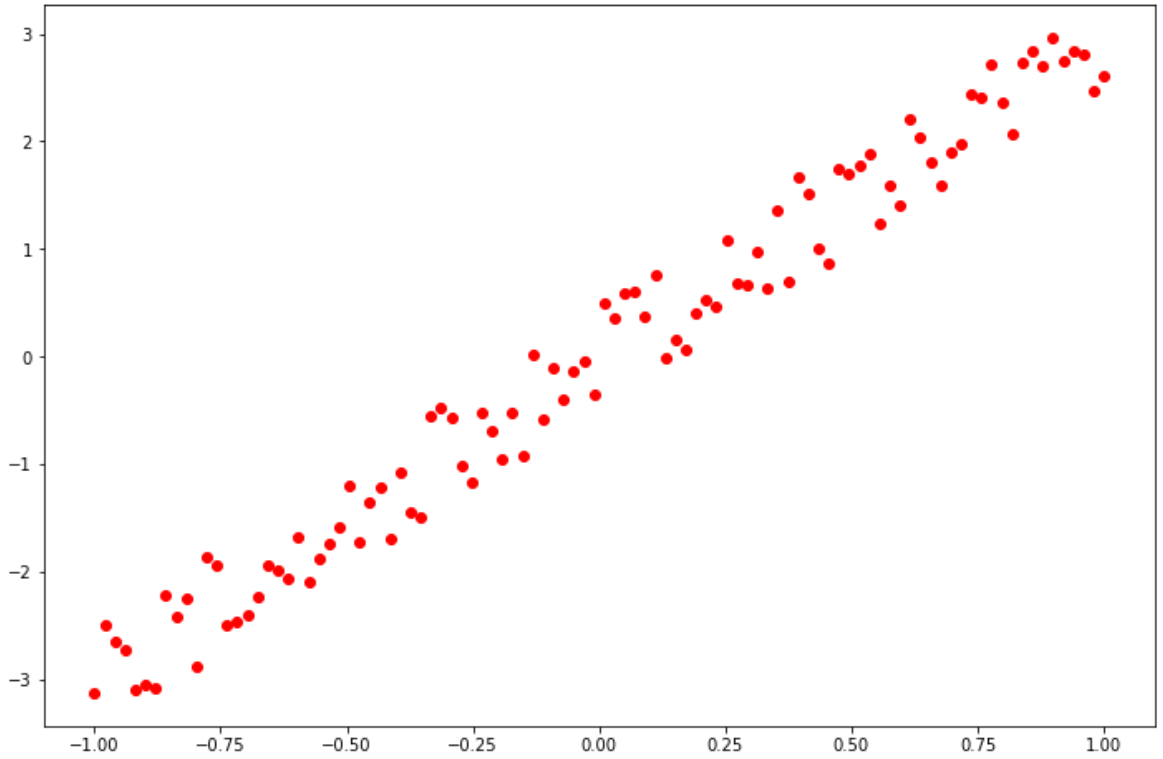
- $\int |F(x) - f(x)|\, dx < \epsilon$

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control Signal Systems* **2,** 303–314 (1989). https://doi.org/10.1007/BF02551274

Entonces… ¿puedo aprender lo que quiera?

Si, pero…

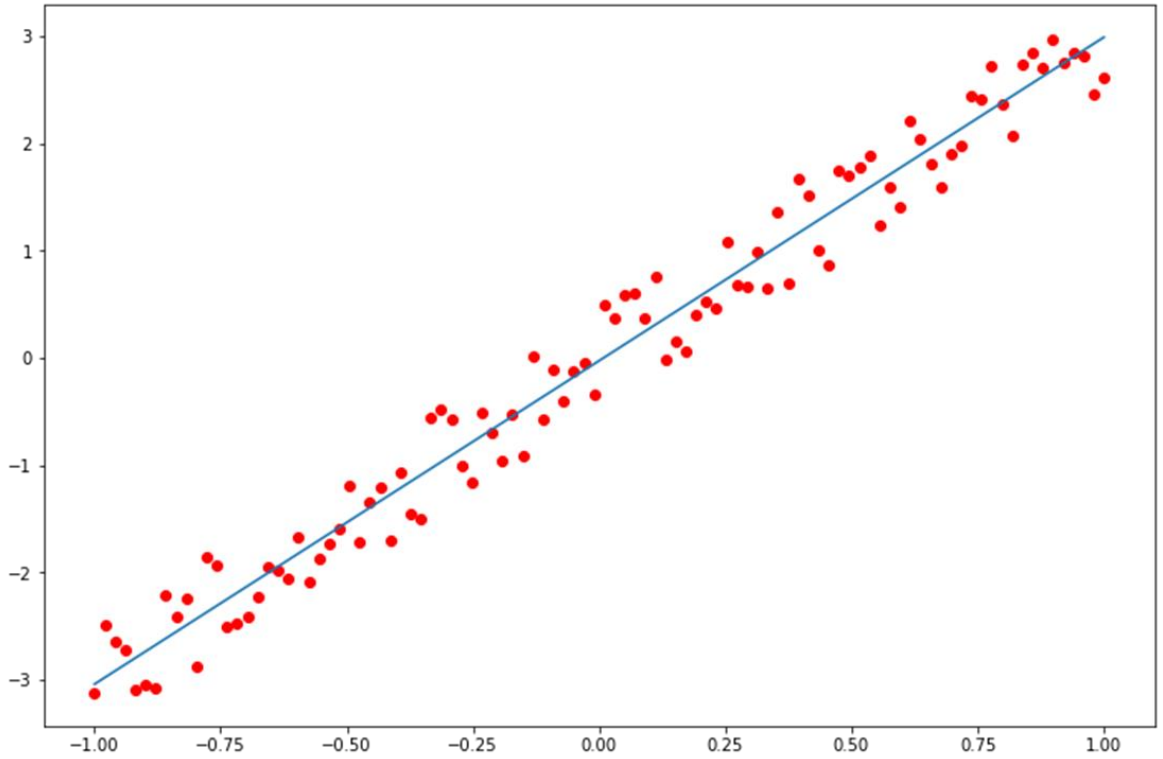
A más capas e hiperparámetros tenga la red, esta es más sensible a pequeñas variaciones en el conjunto de datos de entrenamiento.



Entonces… ¿puedo aprender lo que quiera?

Si, pero…

A más capas e hiperparámetros tenga la red, esta es más sensible a pequeñas variaciones en el conjunto de datos de entrenamiento.

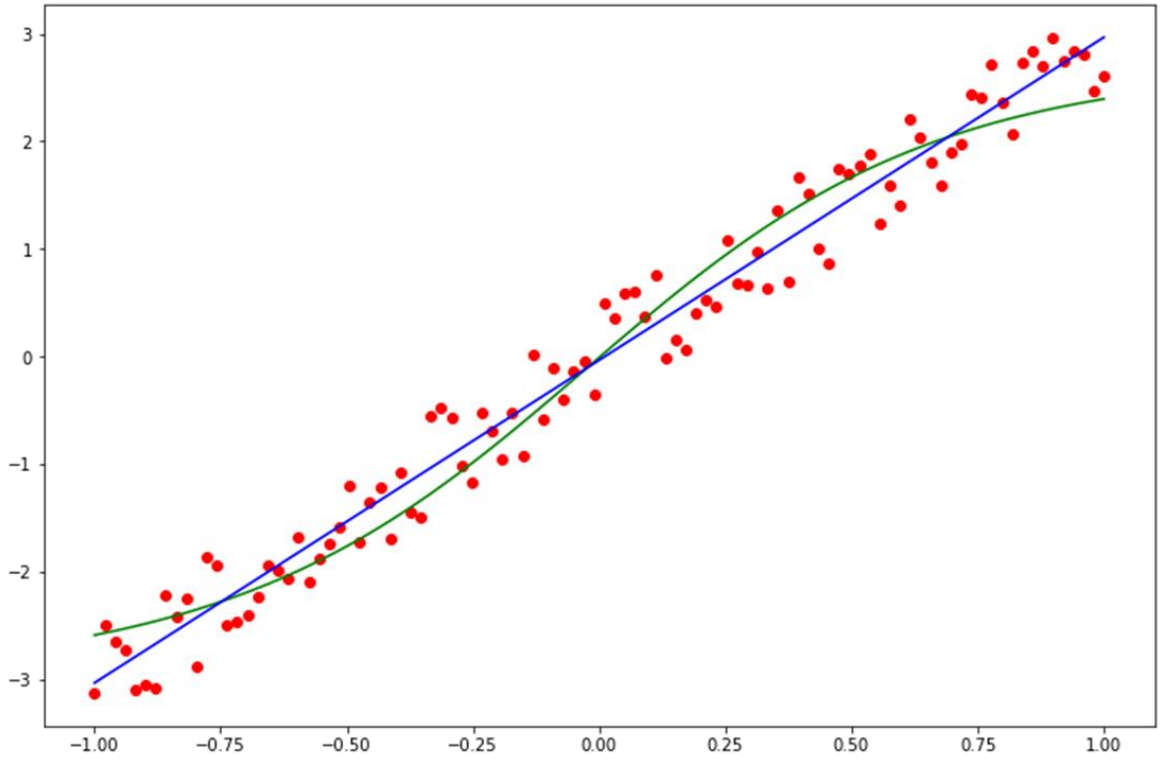



```
i = Input((1,))
d = Dense(1)(i)
model = Model(i, d)
model.compile(loss='mse', optimizer='sgd')
```

Entonces… ¿puedo aprender lo que quiera?

Si, pero…

A más capas e hiperparámetros tenga la red, esta es más sensible a pequeñas variaciones en el conjunto de datos de entrenamiento.




```
i = Input((1,))
d = Dense(2, activation='sigmoid')(i)
d = Dense(1)(d)
model1 = Model(i, d)
model1.compile(loss='mse', optimizer='sgd')
```

Repaso: Overfitting y Undefitting

Bias-Variaze tradeoff

- Underfitting: Error proveniente del algoritmo de aprendizaje. El algoritmo no es capaz de aprender las relaciones entre las entradas y las salidas (El algoritmo tiene alto *bias*).

- Overfitting: El algoritmo aprende el ruido existente en el conjunto de entrenamiento. No generaliza bien para datos en el conjunto de test. (El algoritmo tiene alta *varianze*).

Overfitting

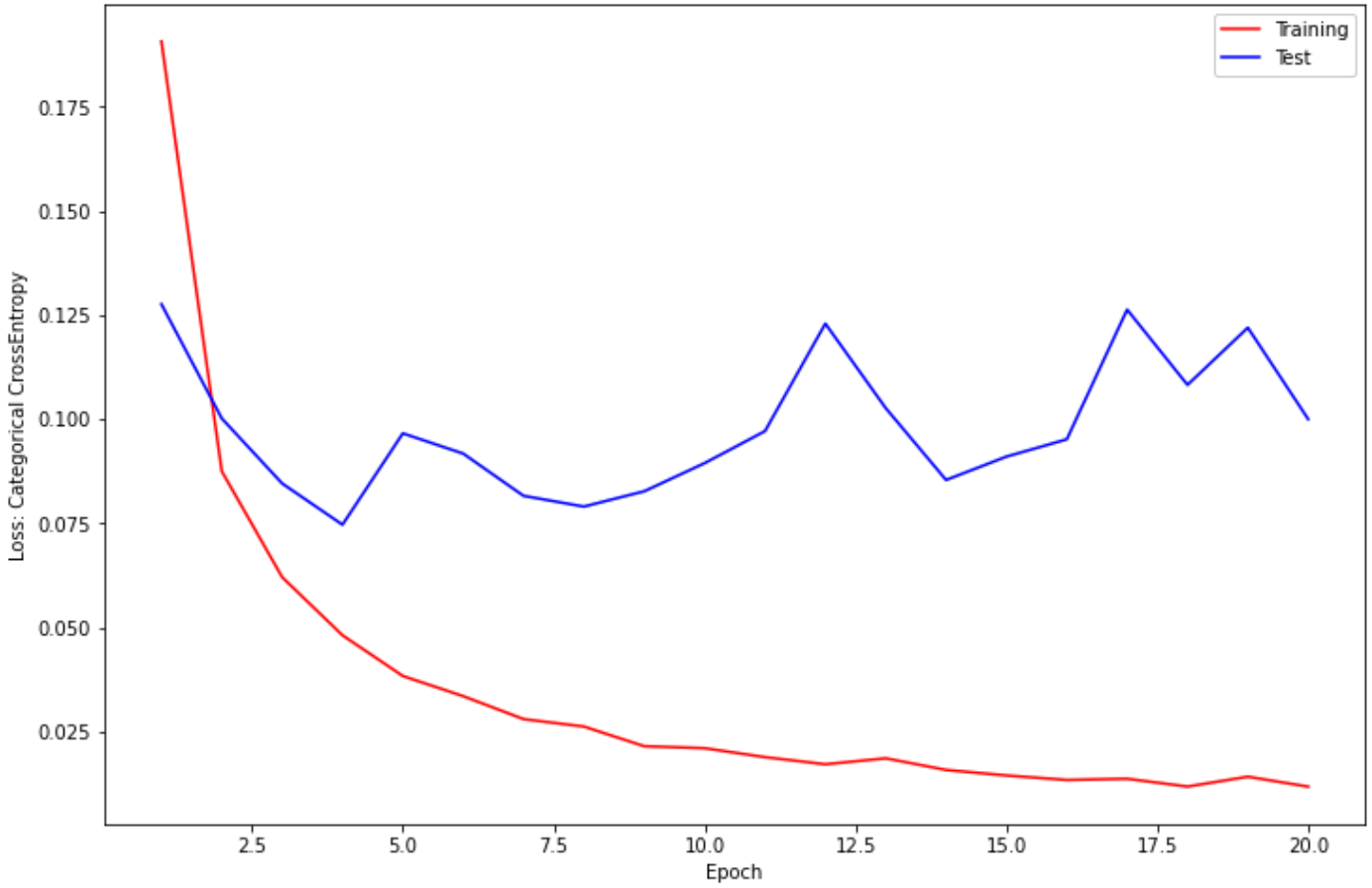
Mnist

Consideremos la siguiente red para clasificar los dígitos en Mnist

```
i = Input((X_train.shape[1],))
d = Dense(512, activation='relu')(i)
d = Dense(256, activation='relu')(d)
d = Dense(128, activation='relu')(d)
d = Dense(10, activation='softmax')(d)
model = Model(i, d)
model.compile(optimizer='adam', loss='categorical_crossentropy')
```

Overfitting

Mnist



Overfitting

¿Qué solución han visto para otro tipo de regresiones (lineales, logísticas) han visto?

Overfitting

Regularizaciones

¿Qué solución han visto para otro tipo de regresiones (lineales, logísticas) han visto?

¡¡REGULARIZACIÓN!!

¿Qué solución han visto para otro tipo de regresiones (lineales, logísticas) han visto?

¡¡REGULARIZACIÓN!!

Básicamente agregar una penalización por pesos “altos” en la función de perdida:

$$L_n(y, \hat{y}, w) = L(y, \hat{y}) + \lambda R(w)$$

¿Qué solución han visto para otro tipo de regresiones (lineales, logísticas) han visto?

¡¡REGULARIZACIÓN!!

Básicamente agregar una penalización por pesos “altos” en la función de perdida:

$$L_n(y, \hat{y}, w) = L(y, \hat{y}) + \lambda R(w)$$

Por ejemplo, regularización L1:

$$L_n(y, \hat{y}, w) = L(y, \hat{y}) + \lambda \sum |w|)$$

¿Qué solución han visto para otro tipo de regresiones (lineales, logísticas) han visto?

¡¡REGULARIZACIÓN!!

Básicamente agregar una penalización por pesos “altos” en la función de perdida:

$$L_n(y, \hat{y}, w) = L(y, \hat{y}) + \lambda R(w)$$

Por ejemplo, regularización L2:

$$L_n(y, \hat{y}, w) = L(y, \hat{y}) + \lambda \sum w^2$$

Overfitting

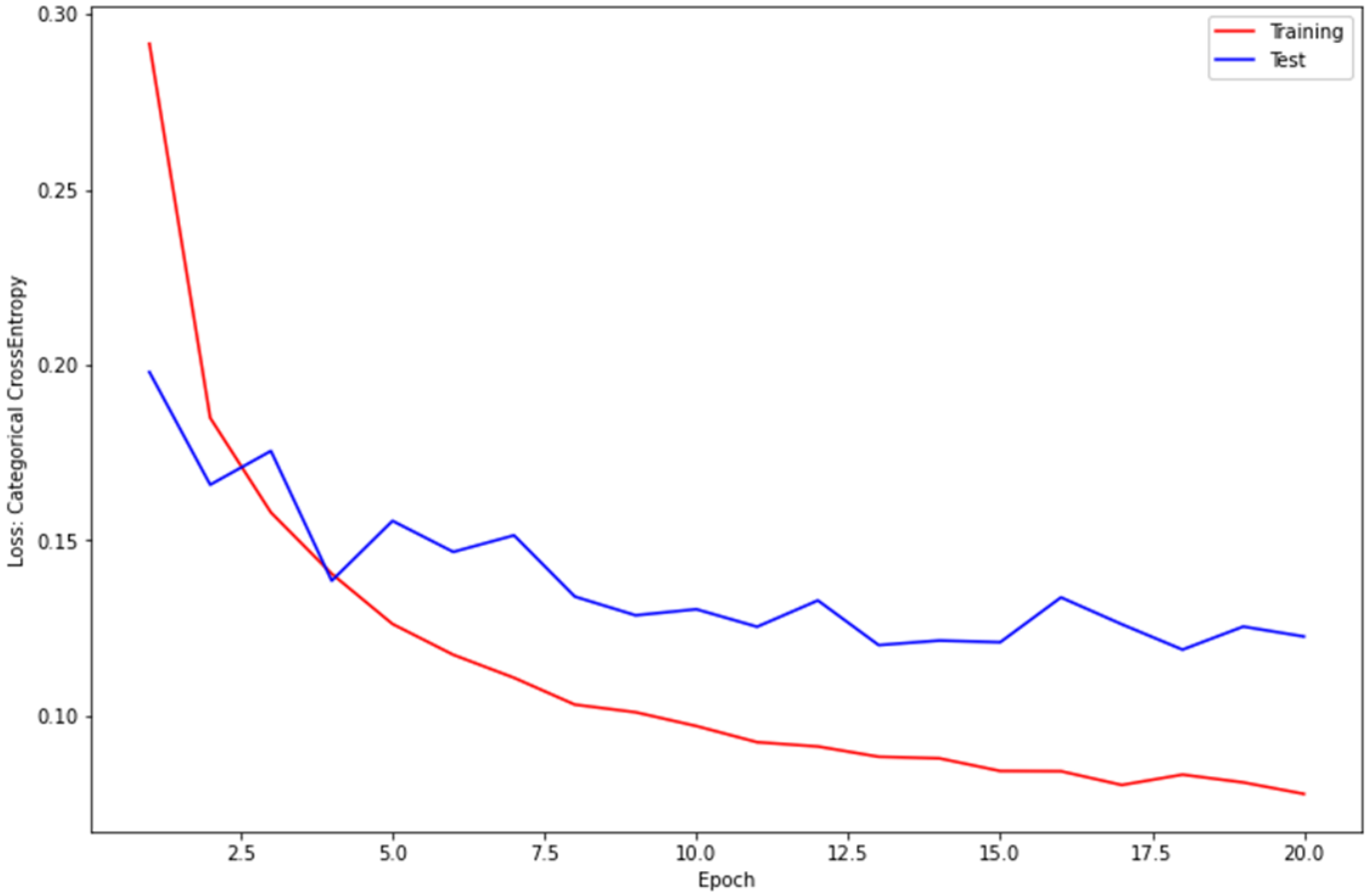
Mnist

Consideremos la siguiente red para clasificar los dígitos en Mnist

```
i = Input((X_train.shape[1],))
d = Dense(512, activation='relu', kernel_regularizer=L2(0.0001))(i)
d = Dense(256, activation='relu', kernel_regularizer=L2(0.0001))(d)
d = Dense(128, activation='relu', kernel_regularizer=L2(0.0001))(d)
d = Dense(10, activation='softmax', kernel_regularizer=L2(0.0001))(d)
model = Model(i, d)
model.compile(optimizer='adam', loss='categorical_crossentropy')
```

Overfitting

Mnist



Overfitting

Noise Layers

Existen diversos tipos de capas que permiten generar ruido entre las distintas capas de una red neuronal para evitar el *overfitting*. Estas capas solo se activan durante el entrenamiento. Las más usadas son:

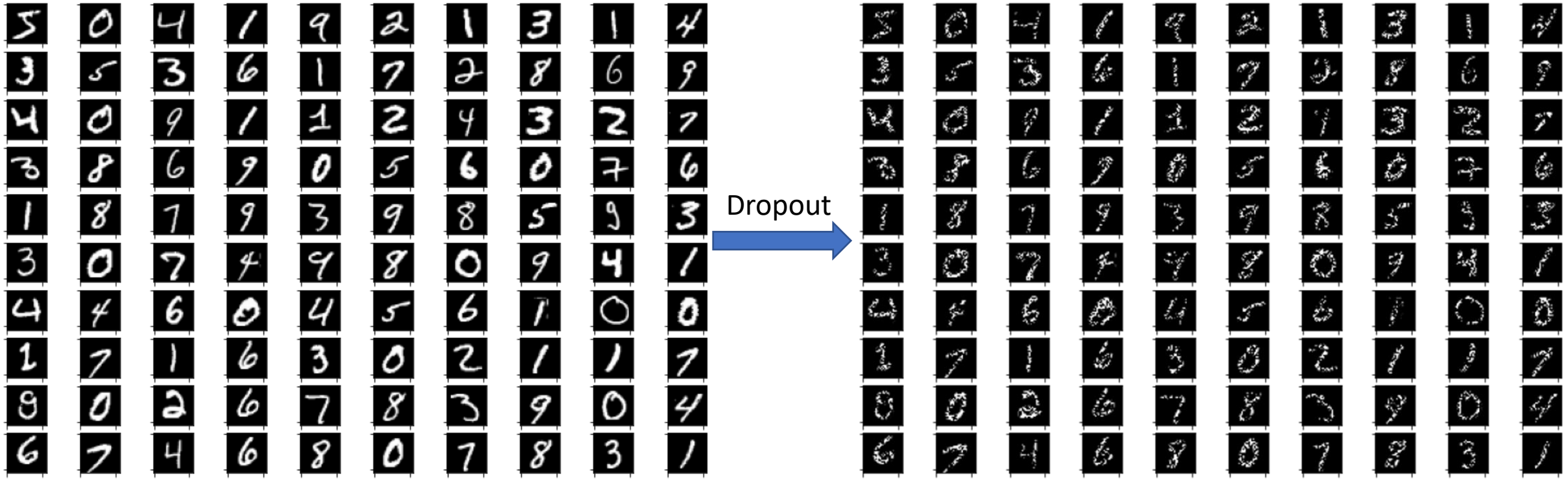
- Dropout: un porcentaje de las entradas van a cero.
- GaussianNoise: le suma a la entrada un ruido de distribución normal, media 0.
- GaussianDropout: multiplica las entradas por una muestra normal de media 1.

También hay capas de normalización que permiten evitar valores extremos o altos:

- BatchNormalization: Normaliza con media 0 y desviación estándar 1 considerando la distribución de los mini-batches.
- LayerNormalization: Normaliza con media 0 y desviación estándar 1 independientemente de los mini-batches.

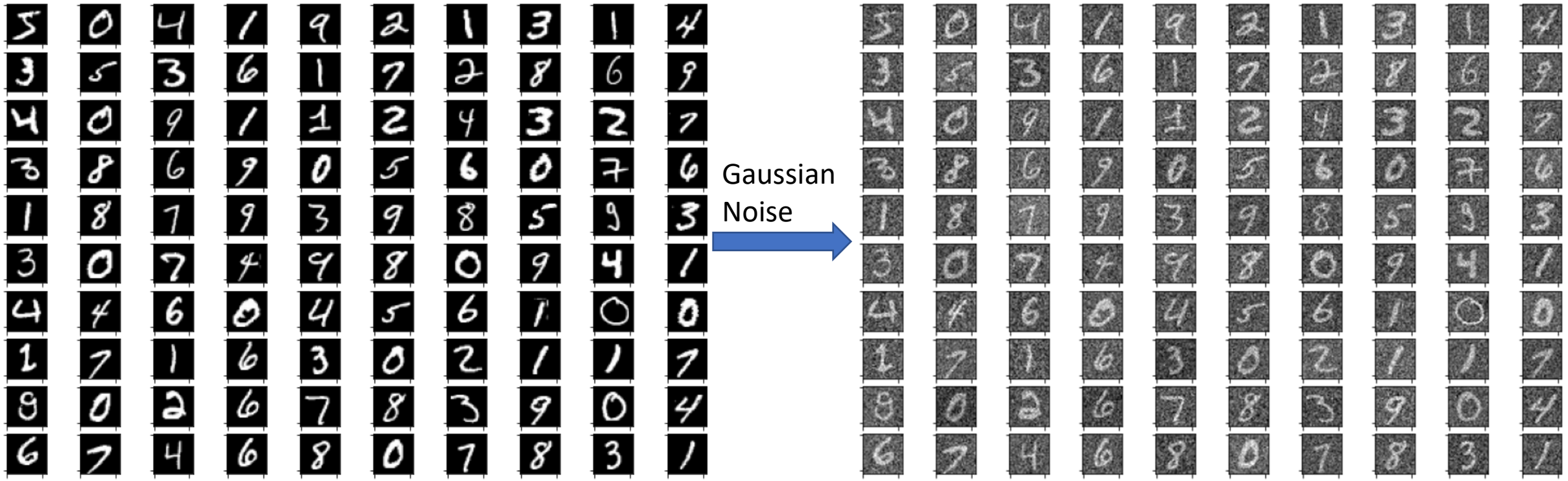
Dropout

Visualizando



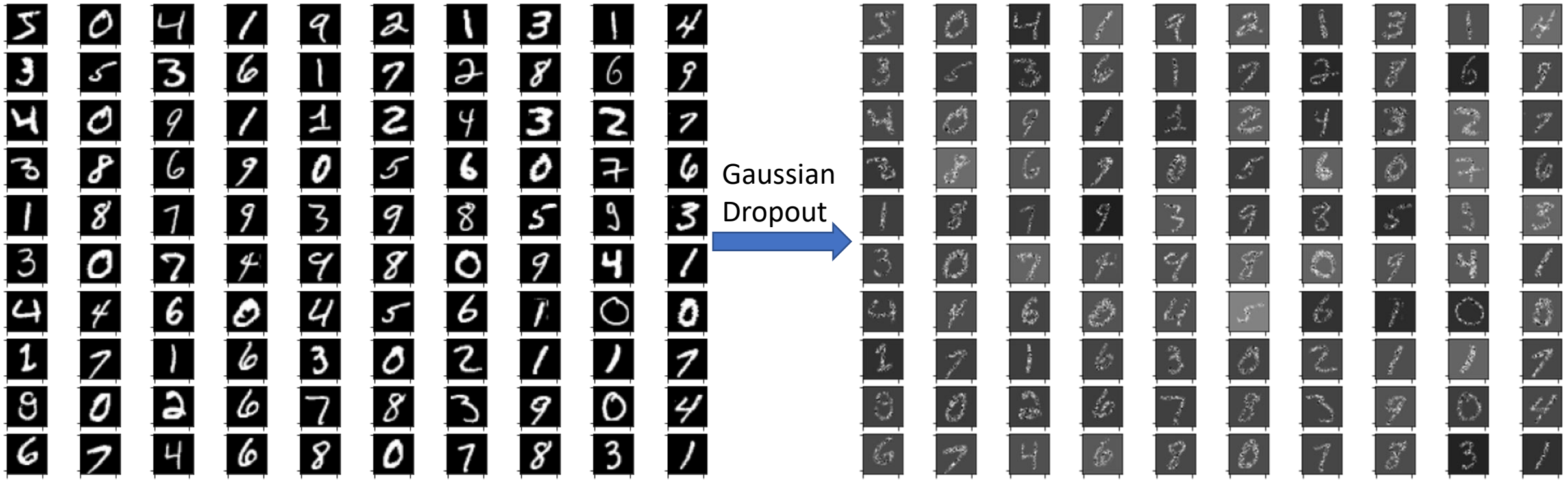
GaussianNoise

Visualizando



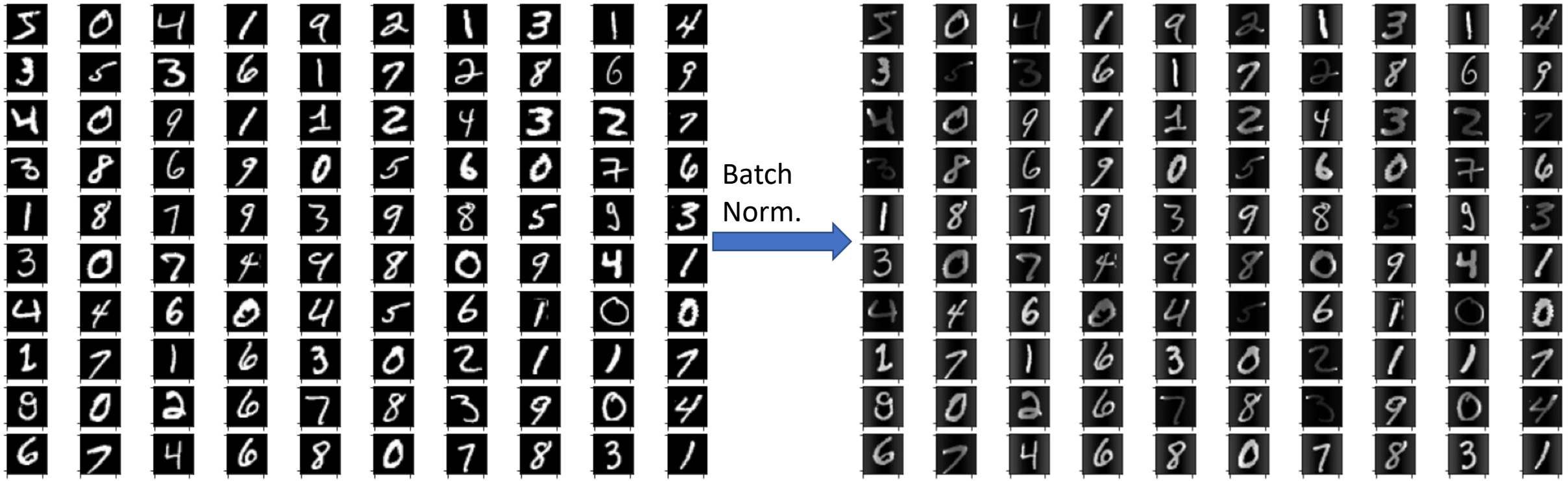
GaussianDropout

Visualizando



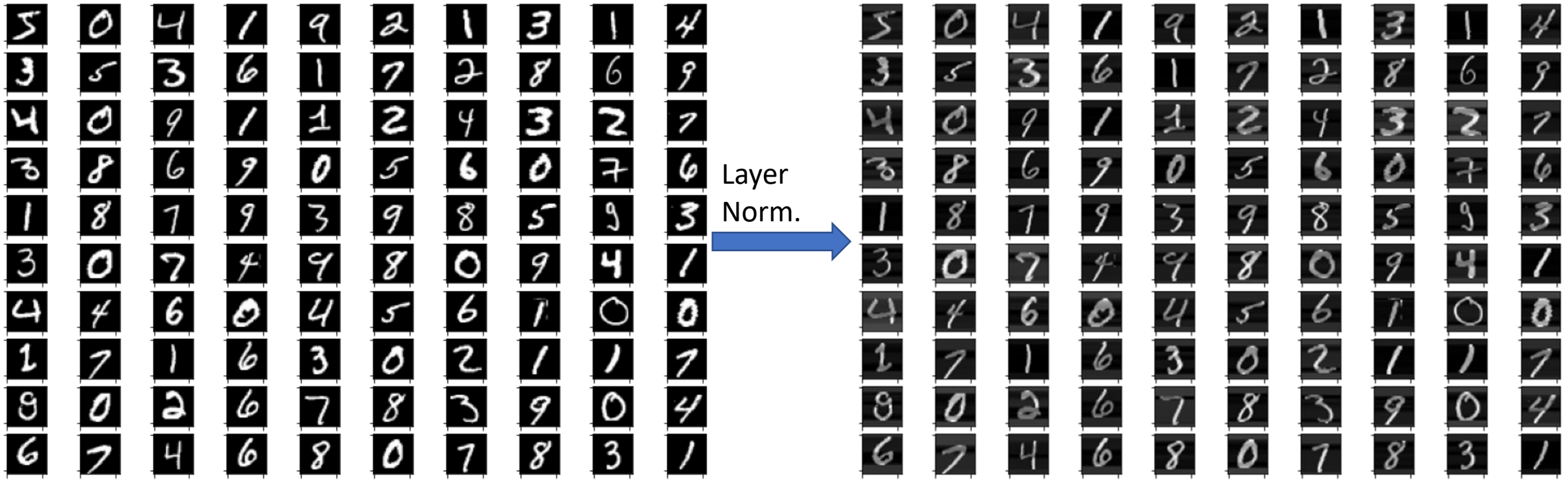
BatchNormalization

Visualizando



LayerNormalization

Visualizando



Overfitting

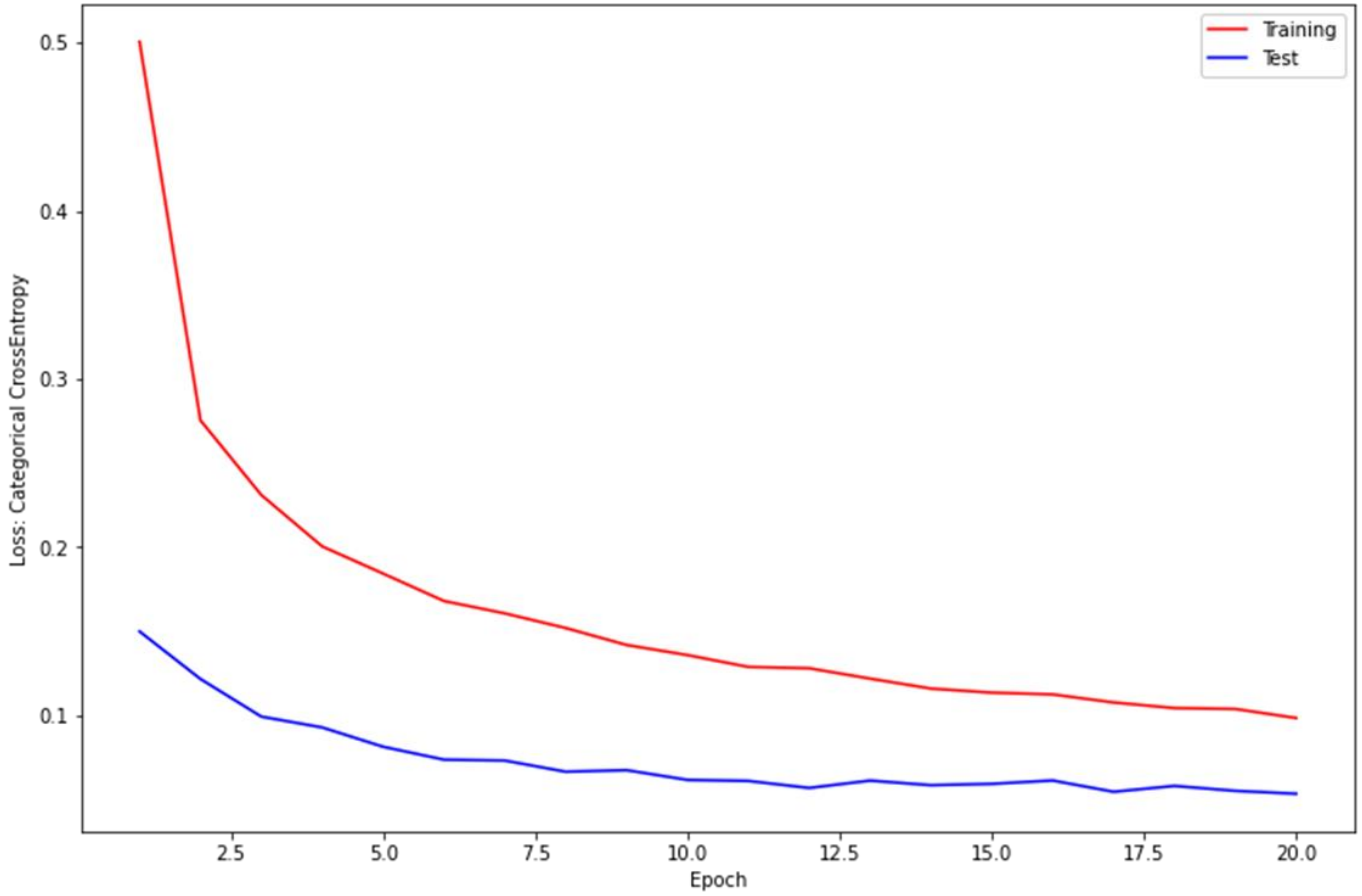
Mnist

Consideremos la siguiente red para clasificar los dígitos en Mnist

```
i = Input((X_train.shape[1],))
d = Dense(512, activation='relu')(i)
d = BatchNormalization()(d)
d = Dropout(0.5)(d)
d = Dense(256, activation='relu')(d)
d = BatchNormalization()(d)
d = Dropout(0.5)(d)
d = Dense(128, activation='relu')(d)
d = BatchNormalization()(d)
d = Dropout(0.5)(d)
d = Dense(10, activation='softmax')(d)
model = Model(i, d)
model.compile(optimizer='adam', loss='categorical_crossentropy')
```

Overfitting

Mnist



EarlyStopping

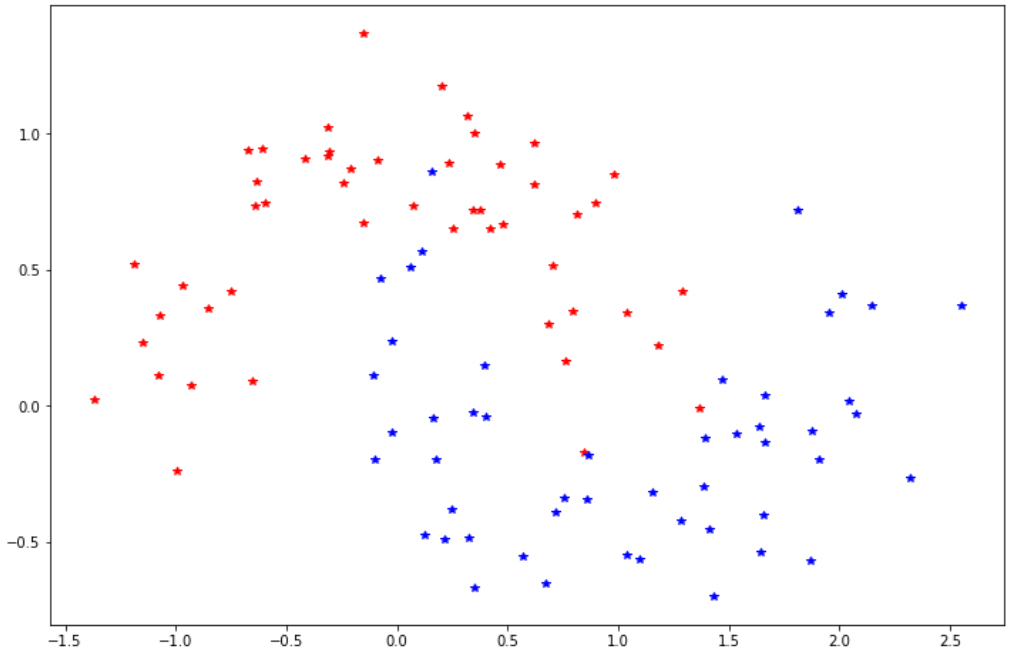
Antes de pasarnos…

- El entrenamiento se basa en varias pasadas sobre el conjunto de entrenamiento, llamadas epochs.
- Al final de cada epoch se espera que el modelo haya mejorado su performance sobre el conjunto de entrenamiento.
- Es difícil definir la cantidad de epoch:
  - Pocos epochs puede hacer que el modelo quede sub-entrenado, es decir, que su performance pueda mejorarse.
  - Muchos epochs pueden hacer que el modelo quede sobre-entrenado (overfitting), es decir, aprende tan bien el conjunto de entrenamiento que generaliza mal.

EarlyStopping

Antes de pasarnos…

Consideremos el siguiente dataset artificial generado con la función make_moons de Sklearn y un potencial modelo.

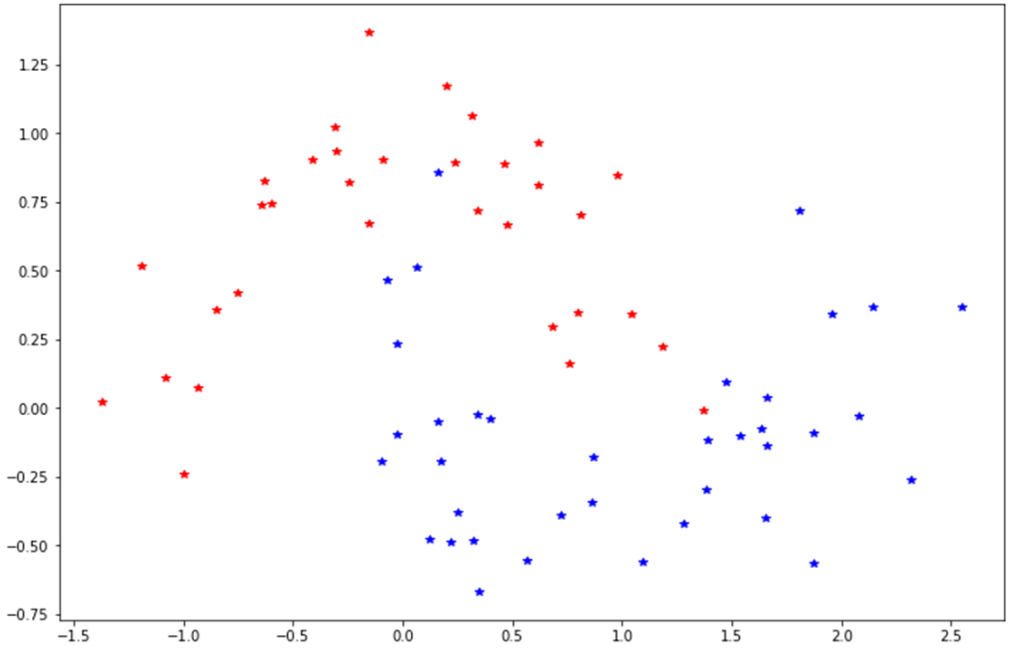


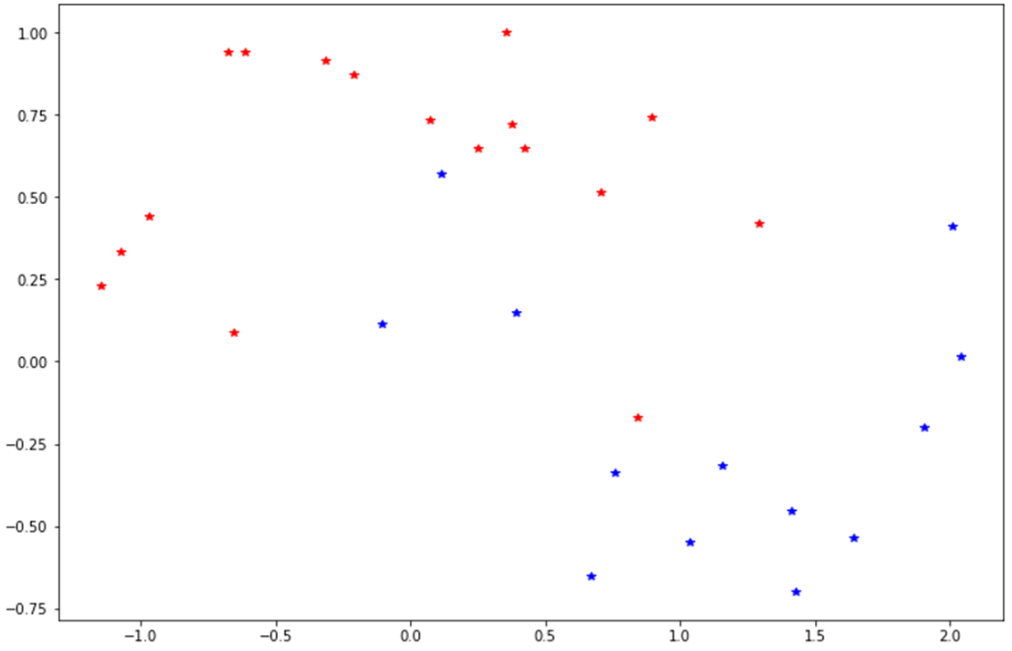

```
i = Input((2,))
d = Dense(500, activation='relu')(i)
d = Dense(1, activation='sigmoid')(d)

model = Model(i, d)
model.compile(loss='binary_crossentropy',
    optimizer='adam', metrics=['accuracy'])
```

EarlyStopping

Antes de pasarnos…

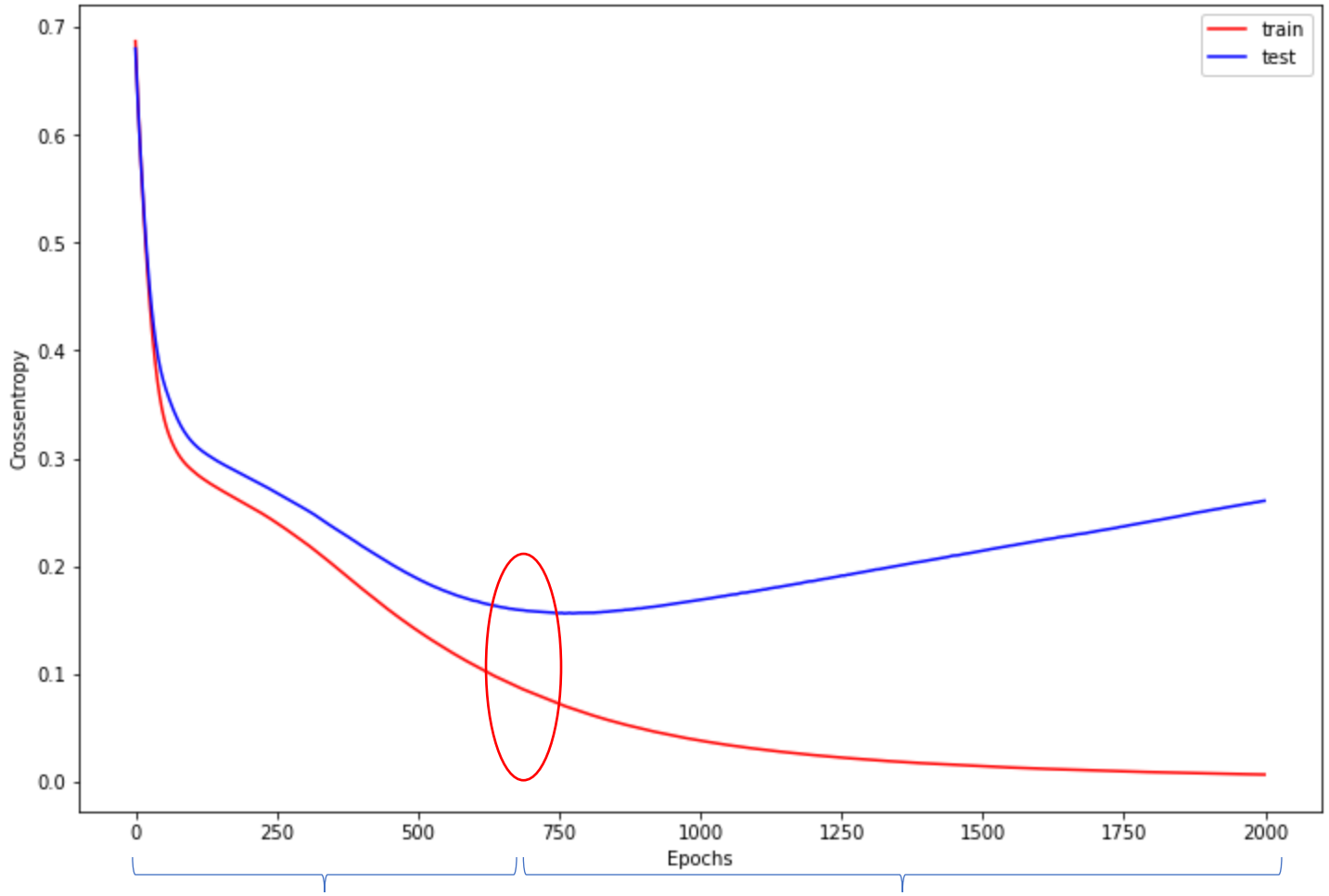
Obviamente, dividimos entre training y test:





EarlyStopping

Antes de pasarnos…



EarlyStopping

Deteniendo el entrenamiento en ese punto mejoramos el resultado para nuestro conjunto de testing.

```
h = model.fit(x_train, y_train, validation_data=(x_test, y_test),
              epochs=2000, verbose=0,
              callbacks=[EarlyStopping('val_loss', mode='min')])
```

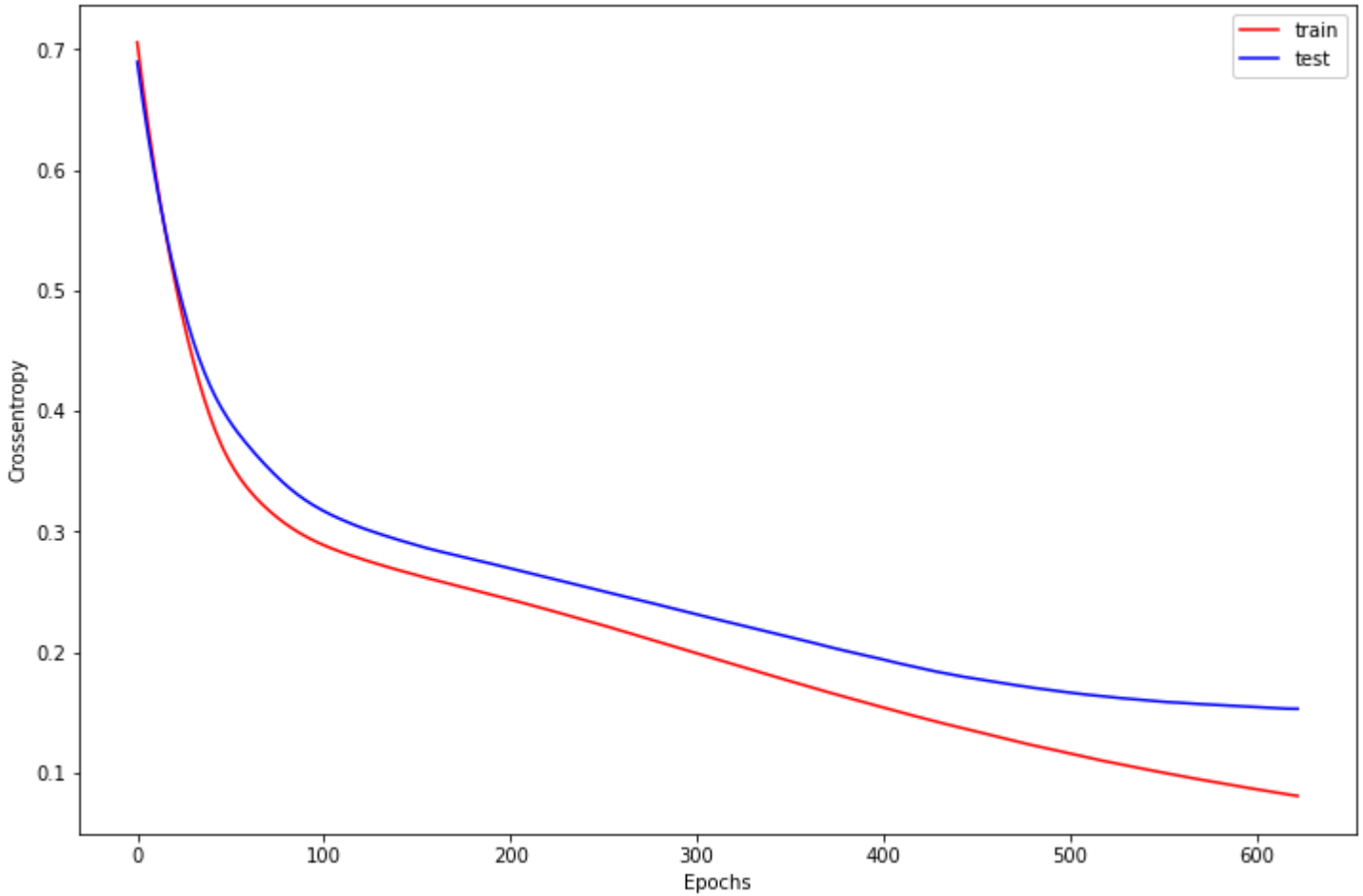
EarlyStopping

Deteniendo el entrenamiento en ese punto mejoramos el resultado para nuestro conjunto de testing.

```
h = model.fit(x_train, y_train, validation_data=(x_test, y_test),
              epochs=2000, verbose=0,
              callbacks=[EarlyStopping('val_loss', mode='min')])
```

- Callbacks se ejecutan en distintos puntos del entrenamiento.
- En particular, EarlyStopping se evalúa al final de cada epoch.

EarlyStopping



EarlyStopping

Deteniendo el entrenamiento en ese punto mejoramos el resultado para nuestro conjunto de testing.

```
h = model.fit(x_train, y_train, validation_data=(x_test, y_test),
              epochs=2000, verbose=0,
              callbacks=[EarlyStopping('val_loss', mode='min')])
```

- Callbacks se ejecutan en distintos puntos del entrenamiento.
- En particular, EarlyStopping se evalúa al final de cada epoch.

IMPORTANTE: Para evaluar el modelo efectivamente se deber usar un tercer conjunto de datos, llamado validación.

Conclusiones

- Las redes neuronales son modelos con una gran capacidad de aprendizaje.
- Estos modelos pueden llegar a “sobre-aprender” lo que se les enseña y no generalizan bien.
- Hay técnicas para mitigar este problema:
  - Regularización de hiperparámetros.
  - Capas de regularización:
    - Dropouts
    - GaussianNoise
    - GaussianDropout
    - ….
  - EarlyStopping