

# INF-477 REDES NEURONALES ARTIFICIALES

## TAREA 1 - PERCEPTRONES MULTICAPA O REDES FF

Gabriel Jara    Francisco Salazar    Daniel San Martín Reyes



UNIVERSIDAD TÉCNICA  
FEDERICO SANTA MARÍA



Departamento de Informática  
Universidad Técnica Federico Santa María

12 de septiembre de 2016



# OUTLINE

1 INTRODUCCIÓN

2 OBJETIVOS

3 DESARROLLO

4 CONCLUSIONES

5 REFERENCIAS



# OUTLINE

## 1 INTRODUCCIÓN

## 2 OBJETIVOS

## 3 DESARROLLO

## 4 CONCLUSIONES

## 5 REFERENCIAS





# REDES NEURONALES ARTIFICIALES

- Motivadas por emular el comportamiento biológico del cerebro humano.
- Sistema altamente complejo, no-lineal y paralelo.
- Capacidad de realizar múltiples operaciones simultáneamente, a diferencia de los computadores del tipo secuencial.
- Una ANN es un procesador de información, de distribución paralela, constituida por unidades pequeñas de procesamiento llamadas neuronas.
- Características.





# REDES NEURONALES ARTIFICIALES

- Motivadas por emular el comportamiento biológico del cerebro humano.
- Sistema altamente complejo, no-lineal y paralelo.
- Capacidad de realizar múltiples operaciones simultáneamente, a diferencia de los computadores del tipo secuencial.
- Una ANN es un procesador de información, de distribución paralela, constituida por unidades pequeñas de procesamiento llamadas neuronas.
- Características.





# REDES NEURONALES ARTIFICIALES

- Motivadas por emular el comportamiento biológico del cerebro humano.
- Sistema altamente complejo, no-lineal y paralelo.
- Capacidad de realizar múltiples operaciones simultáneamente, a diferencia de los computadores del tipo secuencial.
- Una ANN es un procesador de información, de distribución paralela, constituida por unidades pequeñas de procesamiento llamadas neuronas.
- Características.





# REDES NEURONALES ARTIFICIALES

- Motivadas por emular el comportamiento biológico del cerebro humano.
- Sistema altamente complejo, no-lineal y paralelo.
- Capacidad de realizar múltiples operaciones simultáneamente, a diferencia de los computadores del tipo secuencial.
- Una ANN es un procesador de información, de distribución paralela, constituida por unidades pequeñas de procesamiento llamadas neuronas.
- Características.





# REDES NEURONALES ARTIFICIALES

- Motivadas por emular el comportamiento biológico del cerebro humano.
- Sistema altamente complejo, no-lineal y paralelo.
- Capacidad de realizar múltiples operaciones simultáneamente, a diferencia de los computadores del tipo secuencial.
- Una ANN es un procesador de información, de distribución paralela, constituida por unidades pequeñas de procesamiento llamadas neuronas.
- Características.





# OUTLINE

1 INTRODUCCIÓN

2 OBJETIVOS

3 DESARROLLO

4 CONCLUSIONES

5 REFERENCIAS





# OBJETIVOS

- Manipulación de Dataframes usando la librería *Pandas*.
- Estandarización y normalización de datos.
- Manipulación de matrices utilizando *Numpy*.
- Construir perceptrones multicapa utilizando la librería *Keras*.
- Ver métodos de entrenamiento de ANN utilizando *Keras*: Batch, Mini-Batches, Online, Learning Rate, Momentum.
- Validación cruzada - *Cross Validation*, utilizando *sklearn*.



# OBJETIVOS

- Manipulación de Dataframes usando la librería *Pandas*.
- Estandarización y normalización de datos.
- Manipulación de matrices utilizando *Numpy*.
- Construir perceptrones multicapa utilizando la librería *Keras*.
- Ver métodos de entrenamiento de ANN utilizando *Keras*: Batch, Mini-Batches, Online, Learning Rate, Momentum.
- Validación cruzada - *Cross Validation*, utilizando *sklearn*.





# OBJETIVOS

- Manipulación de Dataframes usando la librería *Pandas*.
- Estandarización y normalización de datos.
- Manipulación de matrices utilizando *Numpy*.
- Construir perceptrones multicapa utilizando la librería *Keras*.
- Ver métodos de entrenamiento de ANN utilizando *Keras*: Batch, Mini-Batches, Online, Learning Rate, Momentum.
- Validación cruzada - *Cross Validation*, utilizando *sklearn*.





# OBJETIVOS

- Manipulación de Dataframes usando la librería *Pandas*.
- Estandarización y normalización de datos.
- Manipulación de matrices utilizando *Numpy*.
- Construir perceptrones multicapa utilizando la librería *Keras*.
- Ver métodos de entrenamiento de ANN utilizando *Keras*: Batch, Mini-Batches, Online, Learning Rate, Momentum.
- Validación cruzada - *Cross Validation*, utilizando *sklearn*.



# OBJETIVOS

- Manipulación de Dataframes usando la librería *Pandas*.
- Estandarización y normalización de datos.
- Manipulación de matrices utilizando *Numpy*.
- Construir perceptrones multicapa utilizando la librería *Keras*.
- Ver métodos de entrenamiento de ANN utilizando *Keras*: Batch, Mini-Batches, Online, Learning Rate, Momentum.
- Validación cruzada - *Cross Validation*, utilizando *sklearn*.



# OBJETIVOS

- Manipulación de Dataframes usando la librería *Pandas*.
- Estandarización y normalización de datos.
- Manipulación de matrices utilizando *Numpy*.
- Construir perceptrones multicapa utilizando la librería *Keras*.
- Ver métodos de entrenamiento de ANN utilizando *Keras*: Batch, Mini-Batches, Online, Learning Rate, Momentum.
- Validación cruzada - *Cross Validation*, utilizando *sklearn*.



# OUTLINE

1 INTRODUCCIÓN

2 OBJETIVOS

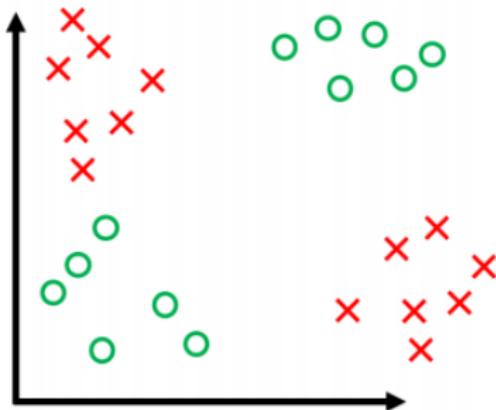
3 DESARROLLO

4 CONCLUSIONES

5 REFERENCIAS



## PREGUNTA 1 - *XOR Problem*



**FIGURA:** Categorización de datos del problema XOR





# XOR Problem

- Generando datos.
- Gráfico.
- Datos linealmente inseparables.
- ¿Resoluble por un perceptrón simple?



# XOR Problem

- Generando datos.
- Gráfico.
- Datos linealmente inseparables.
- ¿Resoluble por un perceptrón simple?

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

n = 1000
rng = np.random.RandomState(0)
X = rng.uniform(low=-1.0,high=1.0,size=(n,2))
Y = np.logical_xor(X[:,0] > 0, X[:,1] > 0)
M = np.c_[X, Y]

df = pd.DataFrame(M, columns=['x', 'y', 'z'])
df.plot.scatter(x='x', y='y', c='z')

plt.show()
```





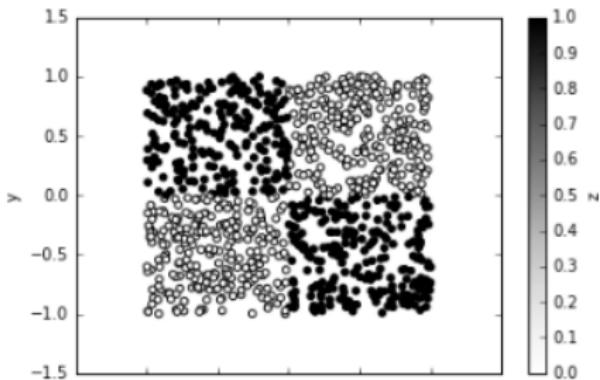
# XOR Problem

- Generando datos.
- Gráfico.
- Datos linealmente inseparables.
- ¿Resoluble por un perceptrón simple?



# XOR Problem

- Generando datos.
- Gráfico.
- Datos linealmente inseparables.
- ¿Resoluble por un perceptrón simple?





# XOR Problem

- Generando datos.
- Gráfico.
- Datos linealmente inseparables.
- ¿Resoluble por un perceptrón simple?





# XOR Problem

- Generando datos.
- Gráfico.
- Datos linealmente inseparables.
- ¿Resoluble por un perceptrón simple?





# XOR Problem

- Resolviendo con una neurona artificial individual.
- *Sin usar librería*
- Gráfico.
- Irresoluble por una sola neurona.
- Necesitamos más capas.
- Perceptrón Multicapa.



# XOR Problem

- Resolviendo con una neurona artificial individual.
- *Sin usar librería*
- Gráfico.
- Irresoluble por una sola neurona.
- Necesitamos más capas.
- Perceptrón Multicapa.

```
from mlxtend.evaluate import plot_decision_regions
from sklearn.linear_model import perceptron
from pandas import *

net = perceptron.Perceptron(n_iter=100, verbose=0, random_state=None,
                             fit_intercept=True, eta_0=0.002)

net.fit(X, Y)

print("Coeficiente 0 " + str(net.coef_[0,0]))
print("Coeficiente 1 " + str(net.coef_[0,1]))
print("Sesgo " + str(net.intercept_[0]))
print("Precisión:" + str(net.score(X, Y)*100) + "%")

plot_decision_regions(X, Y.astype(np.integer), clf=net)
plt.show()
```



# XOR Problem

- Resolviendo con una neurona artificial individual.
- *Sin usar librería*
- Gráfico.
- Irresoluble por una sola neurona.
- Necesitamos más capas.
- Perceptrón Multicapa.



# XOR Problem

- Resolviendo con una neurona artificial individual.
- *Sin usar librería*
- Gráfico.
- Irresoluble por una sola neurona.
- Necesitamos más capas.
- Perceptrón Multicapa.

```
class Perceptron(object):  
  
    def __init__(self, eta=0.02, epochs=100):  
        self.eta = eta  
        self.epochs = epochs  
  
    def train(self, X, y):  
  
        self.w_ = np.zeros(1 + X.shape[1])  
        self.errors_ = []  
  
        for _ in range(self.epochs):  
            errors = 0  
            for xi, yi in zip(X, y):  
                update = self.eta * (yi - self.predict(xi))  
                self.w_[1:] += update * xi  
                self.w_[0] += update  
                errors += int(update != 0.0)  
            self.errors_.append(errors)  
        return self  
  
    def net_input(self, X):  
        return np.dot(X, self.w_[1:]) + self.w_[0]  
  
    def predict(self, X):  
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```



# XOR Problem

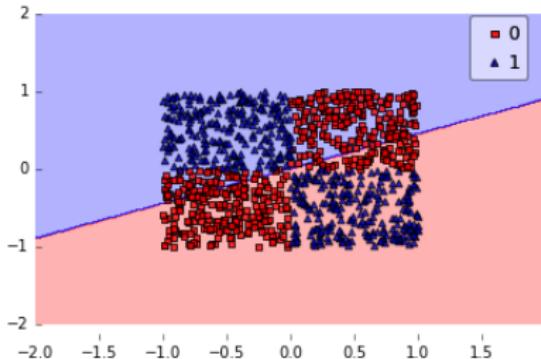
- Resolviendo con una neurona artificial individual.
- *Sin usar librería*
- Gráfico.
- Irresoluble por una sola neurona.
- Necesitamos más capas.
- Perceptrón Multicapa.



# XOR Problem

- Resolviendo con una neurona artificial individual.
- *Sin usar librería*
- Gráfico.
- Irresoluble por una sola neurona.
- Necesitamos más capas.
- Perceptrón Multicapa.

Coeficiente 0 -0.000529204328189  
Coeficiente 1 0.00117731160329  
Sesgo 0.0  
Precisión:49.0%



# XOR Problem

- Resolviendo con una neurona artificial individual.
- *Sin usar librería*
- Gráfico.
- Irresoluble por una sola neurona.
- Necesitamos más capas.
- Perceptrón Multicapa.



# XOR Problem

- Resolviendo con una neurona artificial individual.
- *Sin usar librería*
- Gráfico.
- Irresoluble por una sola neurona.
- Necesitamos más capas.
- Perceptrón Multicapa.

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

model = Sequential()
model.add(Dense(20, input_dim=X.shape[1], init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))

# Compilar modelo
sgd = SGD(lr=0.8)
model.compile(optimizer=sgd, loss='mean_squared_error', metrics=['accuracy'])

# Ajuste de modelo
model.fit(X, Y, nb_epoch=400, verbose=0)

# Evaluar modelo
scores = model.evaluate(X, Y)

print("")
print("Test score: " + str(scores[0]))
print("Test accuracy: " + str(scores[1]))

plot_decision_regions(X, Y.astype(np.integer), clf=model)
plt.show()
```





# XOR Problem

- Resolviendo con una neurona artificial individual.
- *Sin usar librería*
- Gráfico.
- Irresoluble por una sola neurona.
- Necesitamos más capas.
- Perceptrón Multicapa.

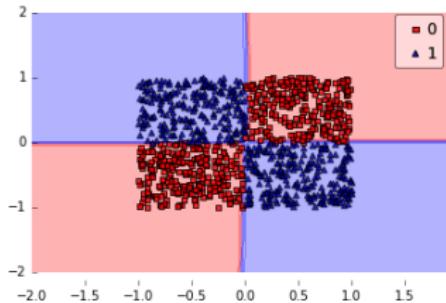


## XOR Problem

- Resolviendo con una neurona artificial individual.
- *Sin usar librería*
- Gráfico.
- Irresoluble por una sola neurona.
- Necesitamos más capas.
- Perceptrón Multicapa.

Using Theano backend.

```
32/1000 [........................] - ETA: 0s
Test score: 0.009803828781
Test accuracy: 0.988
```





## CONCLUSIONES *XOR Problem*

- Una neurona artificial individual sólo puede resolver problemas que son linealmente separables.
- Para problemas no linealmente separables debemos resolverlos utilizando perceptrones multicapa.



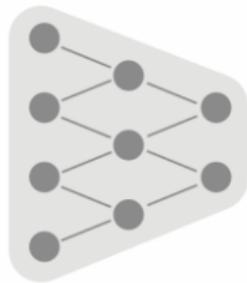


## CONCLUSIONES *XOR Problem*

- Una neurona artificial individual sólo puede resolver problemas que son linealmente separables.
- Para problemas no linealmente separables debemos resolverlos utilizando perceptrones multicapa.



# PREDICCIÓN DEL PRECIO DE UNA CASA



# PREDICCIÓN DEL PRECIO DE UNA CASA - (a) y (b)

- Explicación de las líneas 5 a 9.
- Explicación del Dataset.

```
import pandas as pd
url = 'http://mldata.org/repository/data/download/csv/regression-datasets-housing/'

df = pd.read_csv(url, sep=',', header=None, names=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV'])

from sklearn.cross_validation import train_test_split
df_train, df_test= train_test_split(df, test_size=0.25, random_state=0)
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - (a) y (b)

- Explicación de las líneas 5 a 9.
- Explicación del Dataset.

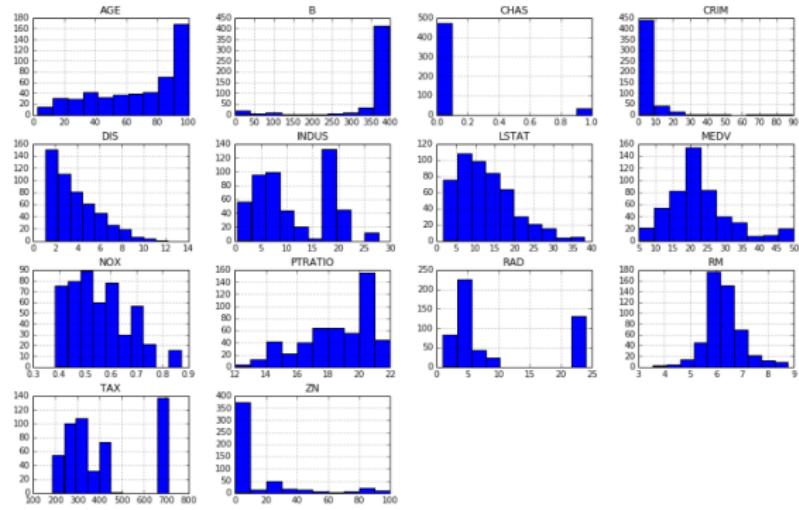
```
df.shape  
df.info()  
df.describe()
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - (a) y (b)

- Explicación de las líneas 5 a 9.
- Explicación del Dataset.

```
df.hist(figsize=(16,10))  
plt.show()
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - (c)

- Normalizar los datos.
- ¿Por qué es importante normalizar los datos?.
- Gradiente descendente - exageración del ajuste.
- Entrenamiento ineficiente ya que de no normalizar se realizarían muchas más iteraciones que normalizando.
- Mayor propensión a quedar atascado en un óptimo local.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (c)

- Normalizar los datos.
- ¿Por qué es importante normalizar los datos?.
- Gradiente descendente - exageración del ajuste.
- Entrenamiento ineficiente ya que de no normalizar se realizarían muchas más iteraciones que normalizando.
- Mayor propensión a quedar atascado en un óptimo local.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(df_train)

# Normalización del conjunto de entrenamiento
X_train_scaled = pd.DataFrame(scaler.transform(df_train), columns=df_train.columns)
y_train_scaled = X_train_scaled.pop('MEDV')

# Normalización del conjunto de testing
X_test_scaled = pd.DataFrame(scaler.transform(df_test), columns=df_test.columns)
y_test_scaled = X_test_scaled.pop('MEDV')
```

# PREDICCIÓN DEL PRECIO DE UNA CASA - (c)

- Normalizar los datos.
- ¿Por qué es importante normalizar los datos?.
- Gradiente descendente - exageración del ajuste.
- Entrenamiento ineficiente ya que de no normalizar se realizarían muchas más iteraciones que normalizando.
- Mayor propensión a quedar atascado en un óptimo local.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (c)

- Normalizar los datos.
- ¿Por qué es importante normalizar los datos?.
- Gradiente descendente - exageración del ajuste.
- Entrenamiento ineficiente ya que de no normalizar se realizarían muchas más iteraciones que normalizando.
- Mayor propensión a quedar atascado en un óptimo local.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (c)

- Normalizar los datos.
- ¿Por qué es importante normalizar los datos?.
- Gradiente descendente - exageración del ajuste.
- Entrenamiento ineficiente ya que de no normalizar se realizarían muchas más iteraciones que normalizando.
- Mayor propensión a quedar atascado en un óptimo local.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (c)

- Normalizar los datos.
- ¿Por qué es importante normalizar los datos?.
- Gradiente descendente - exageración del ajuste.
- Entrenamiento ineficiente ya que de no normalizar se realizarían muchas más iteraciones que normalizando.
- Mayor propensión a quedar atascado en un óptimo local.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(d)$

- Error cuadrático medio vs número de epochs de entrenamiento.
- Entrenamiento utilizando Gradiente descendente estocástico.
- El proceso de aprendizaje no converge.
- El error tiende a infinito desde la primera iteración.
- Depende de la magnitud del *learning rate*.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (d)

- Error cuadrático medio vs número de epochs de entrenamiento.
- Entrenamiento utilizando Gradiente descendente estocástico.
- El proceso de aprendizaje no converge.
- El error tiende a infinito desde la primera iteración.
- Depende de la magnitud del *learning rate*.

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

model = Sequential()
model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform'))
model.add(Activation('sigmoid'))
model.add(Dense(1, init='uniform'))
model.add(Activation('linear'))

sgd = SGD(lr=0.2)
model.compile(optimizer=sgd, loss='mean_squared_error')

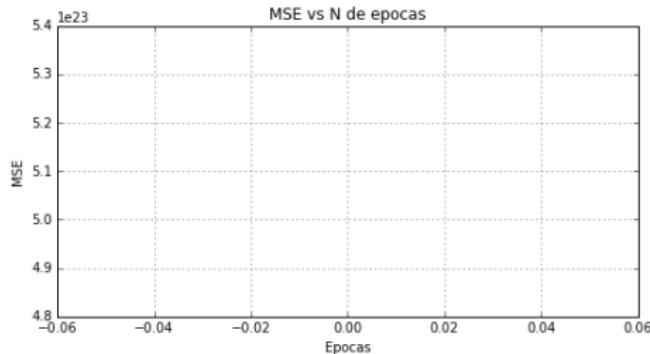
hist1 = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), nb_epoch=300, verbose=0
, validation_data=(X_test_scaled.as_matrix(), y_test_scaled.as_matrix()))

plt.figure(figsize=(8, 4))
plt.plot(hist1.epoch, hist1.history['loss'])
plt.title("MSE vs N de épocas")
plt.xlabel("Epochs")
plt.ylabel("MSE")
plt.grid(True)
plt.show()
print("Último MSE: " + str(hist1.history['loss'][-1]))
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(d)$

- Error cuadrático medio vs número de epochs de entrenamiento.
- Entrenamiento utilizando Gradiente descendente estocástico.
- El proceso de aprendizaje no converge.
- El error tiende a infinito desde la primera iteración.
- Depende de la magnitud del *learning rate*.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(d)$

- Error cuadrático medio vs número de epochs de entrenamiento.
- Entrenamiento utilizando Gradiente descendente estocástico.
- El proceso de aprendizaje no converge.
- El error tiende a infinito desde la primera iteración.
- Depende de la magnitud del *learning rate*.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (d)

- Error cuadrático medio vs número de epochs de entrenamiento.
- Entrenamiento utilizando Gradiente descendente estocástico.
- El proceso de aprendizaje no converge.
- El error tiende a infinito desde la primera iteración.
- Depende de la magnitud del *learning rate*.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (d)

- Error cuadrático medio vs número de epochs de entrenamiento.
- Entrenamiento utilizando Gradiente descendente estocástico.
- El proceso de aprendizaje no converge.
- El error tiende a infinito desde la primera iteración.
- Depende de la magnitud del *learning rate*.





# PREDICCIÓN DEL PRECIO DE UNA CASA - (e)

- Utilizando *ReLU*.
- Con  $learning\ rate = 0,2$  y función de activación *ReLU*, el método converge.
- Beneficios de utilizar *ReLU*
- Posible señal de *Overfitting*



# PREDICCIÓN DEL PRECIO DE UNA CASA - (e)

- Utilizando *RELU*.
- Con  $learning\ rate = 0,2$  y función de activación *RELU*, el método converge.
- Beneficios de utilizar *RELU*
- Posible señal de *Overfitting*

```
model = Sequential()
model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='linear'))

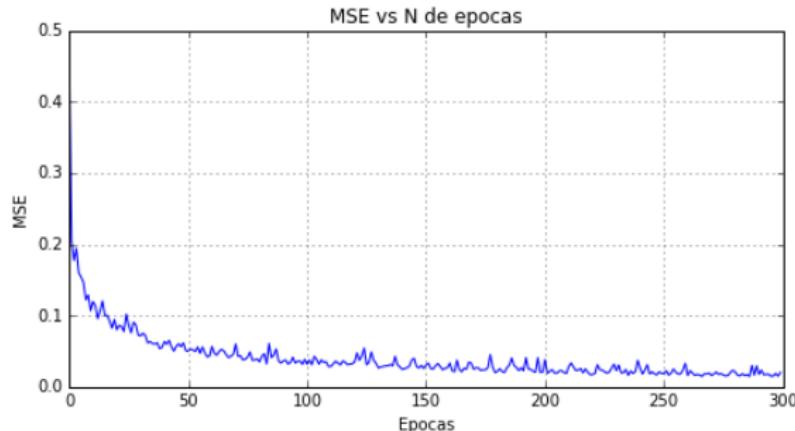
sgd = SGD(lr=0.2)
model.compile(optimizer=sgd, loss='mean_squared_error')

hist2 = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), nb_epoch=300, verbose=0,
 , validation_data=(X_test_scaled.as_matrix(), y_test_scaled.as_matrix()))
```

```
plt.figure(figsize=(8, 4))
plt.plot(hist2.epoch, hist2.history['loss'])
plt.title("MSE vs N de épocas")
plt.xlabel("Épocas")
plt.ylabel("MSE")
plt.grid(True)
plt.show()
print("Último MSE: " + str((hist2.history['loss'])[-1]))
```

# PREDICCIÓN DEL PRECIO DE UNA CASA - (e)

- Utilizando *RELU*.
- Con  $learning\ rate = 0,2$  y función de activación *RELU*, el método converge.
- Beneficios de utilizar *RELU*
- Posible señal de *Overfitting*



# PREDICCIÓN DEL PRECIO DE UNA CASA - (e)

- Utilizando *ReLU*.
- Con  $learning\ rate = 0,2$  y función de activación *ReLU*, el método converge.
- Beneficios de utilizar *ReLU*
- Posible señal de *Overfitting*



# PREDICCIÓN DEL PRECIO DE UNA CASA - (e)

- Utilizando *ReLU*.
- Con  $learning\ rate = 0,2$  y función de activación *ReLU*, el método converge.
- Beneficios de utilizar *ReLU*
- Posible señal de *Overfitting*



# PREDICCIÓN DEL PRECIO DE UNA CASA - (e)

- Utilizando *ReLU*.
- Con  $learning\ rate = 0,2$  y función de activación *ReLU*, el método converge.
- Beneficios de utilizar *ReLU*
- Posible señal de *Overfitting*



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(f)$

- Repetimos el proceso, esta vez variando el *learning rate*.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (*f*)

- Repetimos el proceso, esta vez variando el *learning rate*.

```
import numpy as np

n_lr = 21
lear_rate = np.linspace(0,0.2,n_lr)

plt.figure(figsize=(14, 10))

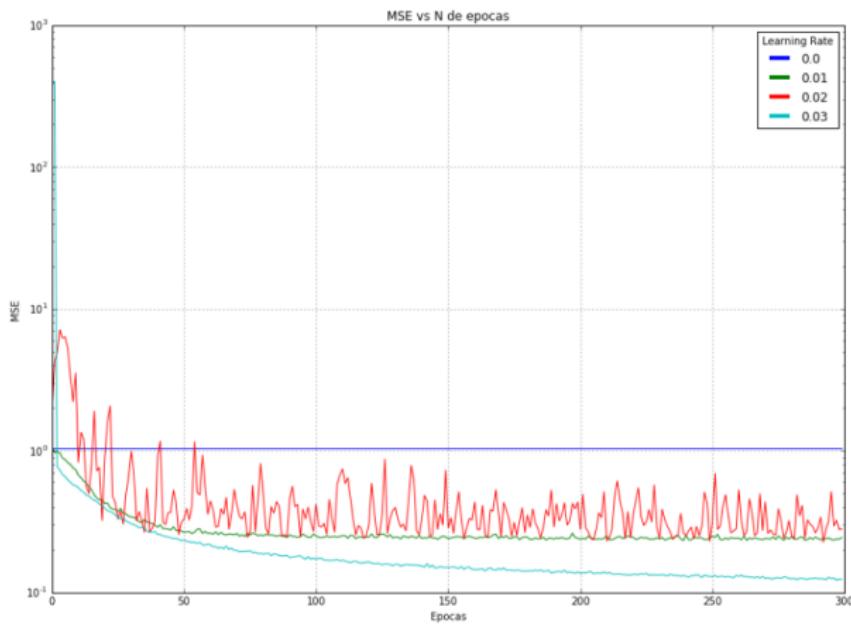
for i in range(n_lr):
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform', activation='sigmoid'))
    model.add(Dense(1, init='uniform', activation='linear'))
    sgd = SGD(lr=lear_rate[i])
    model.compile(optimizer=sgd, loss='mean_squared_error')
    hist = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), nb_epoch=300, verbose=0, validation_data=(X_test_scaled.as_matrix(), y_test_scaled.as_matrix()))
    print("Learning Rate: "+str(round(lear_rate[i],3)) + ", Ultimo MSE de Entrenamiento: " + str(round(hist.history['loss'][-1], 3)))
    #if (i+1) % 4 == 0:
    #    plt.loglog(hist.epoch, hist.history['loss'], label=str(round(lear_rate[i], 3)))
    if np.isnan(hist.history['loss'][-1]) == False:
        plt.semilogy(hist.epoch, hist.history['loss'], label=str(round(lear_rate[i], 3)))

plt.title("MSE vs N de épocas")
plt.xlabel("Epocas")
plt.ylabel("MSE")
plt.grid(True)
leg = plt.legend(title="Learning Rate")
# Ancho de líneas en legend
for legobj in leg.legendHandles:
    legobj.set_linewidth(4.0)
plt.show()
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(f)$

- Repetimos el proceso, esta vez variando el *learning rate*.





# PREDICCIÓN DEL PRECIO DE UNA CASA - ( $f$ )

- Se estima conveniente hacer variar el *learning rate* entre 0 y 0.2
- Se observa que los 3 primeros intentos con *learning rate* = 0.3 fueron convergentes.
- Alta sensibilidad del modelo con respecto al valor del *learning rate*





# PREDICCIÓN DEL PRECIO DE UNA CASA - ( $f$ )

- Se estima conveniente hacer variar el *learning rate* entre 0 y 0.2
- Se observa que los 3 primeros intentos con *learning rate* = 0.3 fueron convergentes.
- Alta sensibilidad del modelo con respecto al valor del *learning rate*





# PREDICCIÓN DEL PRECIO DE UNA CASA - ( $f$ )

- Se estima conveniente hacer variar el *learning rate* entre 0 y 0.2
- Se observa que los 3 primeros intentos con *learning rate* = 0.3 fueron convergentes.
- Alta sensibilidad del modelo con respecto al valor del *learning rate*



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(f)$

- Repetimos el proceso, con *learning rate* entre 0 y 0.04.



# PREDICCIÓN DEL PRECIO DE UNA CASA - ( $f$ )

- Repetimos el proceso, con *learning rate* entre 0 y 0.04.

```
import numpy as np

n_lr = 8
lear_rate = np.linspace(0,0.04,n_lr)

plt.figure(figsize=(14, 10))

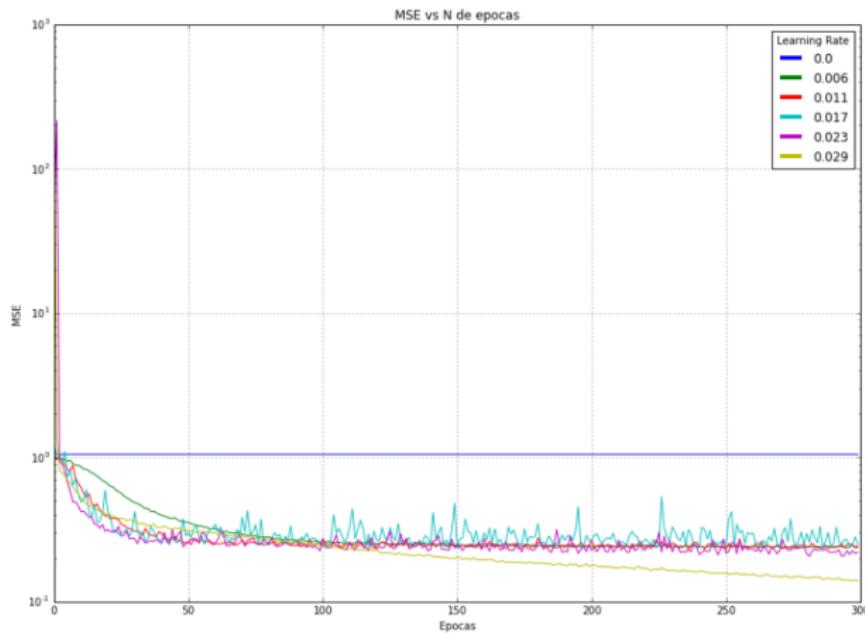
for i in range(n_lr):
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform', activation='sigmoid'))
    model.add(Dense(1, init='uniform', activation='linear'))
    sgd = SGD(lr=lear_rate[i])
    model.compile(optimizer=sgd, loss='mean_squared_error')
    hist = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), nb_epoch=300, verbose=0, validation_data=(X_test_scaled.as_matrix(), y_test_scaled.as_matrix()))
    print("Learning Rate: "+str(round(lear_rate[i],3)) + ", Ultimo MSE de Entrenamiento: " + str(round(hist.history['loss'][-1], 3)))
    #if (i+1) % 4 == 0:
    #    plt.loglog(hist.epoch, hist.history['loss'], label=str(round(lear_rate[i], 3)))
    if np.isnan(hist.history['loss'][-1]) == False:
        plt.semilogy(hist.epoch, hist.history['loss'], label=str(round(lear_rate[i], 3)))

plt.title("MSE vs N de épocas")
plt.xlabel("Epocas")
plt.ylabel("MSE")
plt.grid(True)
leg = plt.legend(title="Learning Rate")
# Ancho de líneas en legend
for legobj in leg.legendHandles:
    legobj.set_linewidth(4.0)
plt.show()
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(f)$

- Repetimos el proceso, con *learning rate* entre 0 y 0.04.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(f)$

- Comportamiento observado anteriormente se repite.
- Posible señal de *overfitting*.
- La mayoría converge en un error cuadrático en torno a 0,23.
- Mientras más alto el *learning rate* más alto el error.
- Riesgo de divergencia.



# PREDICCIÓN DEL PRECIO DE UNA CASA - ( $f$ )

- Comportamiento observado anteriormente se repite.
- Posible señal de *overfitting*.
- La mayoría converge en un error cuadrático en torno a 0,23.
- Mientras más alto el *learning rate* más alto el error.
- Riesgo de divergencia.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(f)$

- Comportamiento observado anteriormente se repite.
- Posible señal de *overfitting*.
- La mayoría converge en un error cuadrático en torno a 0,23.
- Mientras más alto el *learning rate* más alto el error.
- Riesgo de divergencia.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(f)$

- Comportamiento observado anteriormente se repite.
- Posible señal de *overfitting*.
- La mayoría converge en un error cuadrático en torno a 0,23.
- Mientras más alto el *learning rate* más alto el error.
- Riesgo de divergencia.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(f)$

- Comportamiento observado anteriormente se repite.
- Posible señal de *overfitting*.
- La mayoría converge en un error cuadrático en torno a 0,23.
- Mientras más alto el *learning rate* más alto el error.
- Riesgo de divergencia.



# PREDICCIÓN DEL PRECIO DE UNA CASA - ( $g$ )

- Estimación del error de predicción de los modelos usando:.
- Validación cruzada con  $K=5$  y  $k=10$ .



# PREDICCIÓN DEL PRECIO DE UNA CASA - ( $g$ )

- Estimación del error de predicción de los modelos usando:.
- Validación cruzada con  $K=5$  y  $k=10$ .



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(g)$

- Estimación del error de predicción de los modelos usando:.
- Validación cruzada con  $K=5$  y  $k=10$ .

```
from sklearn import cross_validation
Xm = X_train_scaled.as_matrix()
ym = y_train_scaled.as_matrix()

kfold = cross_validation.KFold(len(Xm), 5)
cvscores = []
for i, (train, val) in enumerate(kfold):
    # create model
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform', activation='sigmoid'))
    model.add(Dense(1, init='uniform'))
    model.add(Activation('linear'))
    # Compile model
    sgd = SGD(lr=0.023)
    model.compile(optimizer=sgd, loss='mean_squared_error')
    # Fit the model
    model.fit(Xm[train], ym[train], nb_epoch=300, verbose=0)
    # evaluate the model
    scores = model.evaluate(Xm[val], ym[val])
    cvscores.append(scores)

mse_cv = np.mean(cvscores)
print(", MSE con K=5: " + str(mse_cv))
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - ( $g$ )

- Estimación del error de predicción de los modelos usando:.
- Validación cruzada con  $K=5$  y  $k=10$ .

```
kfold = cross_validation.KFold(len(Xm), 10)
cvscores = []
for i, (train, val) in enumerate(kfold):
    # create model
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform', activation='sigmoid'))
    model.add(Dense(1, init='uniform'))
    model.add(Activation('linear'))
    # Compile model
    sgd = SGD(lr=0.023)
    model.compile(optimizer=sgd, loss='mean_squared_error')
    # Fit the model
    model.fit(Xm[train], ym[train], nb_epoch=300, verbose=0)
    # evaluate the model
    scores = model.evaluate(Xm[val], ym[val])
    cvscores.append(scores)

mse_cv = np.mean(cvscores)
print("MSE con K=10: " + str(mse_cv))

32/75 [=====>.....] - ETA: 0s, MSE con K=5: 0.262823812169
32/37 [=====>.....] - ETA: 0s, MSE con K=10: 0.265327280957
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(g)$

- Se utilizó un valor distinto a 0.2 para el *learning rate*
- Con *learning rate* de 0.023 y utilizando *Sigmoid* se alcanzaba un error cuadrático medio de 0.22 luego de 300 iteraciones en los experimentos realizados en la pregunta  $(f)$ .
- Al utilizar *Cross Validation* el error aumenta a 0.26 para  $K=5$  y  $K=10$ .
- Se puede esperar que para nuevos datos se estime el valor efectivo con un error de 0.26 en lugar de los 0.22 anteriores.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (g)

- Se utilizó un valor distinto a 0.2 para el *learning rate*
- Con *learning rate* de 0.023 y utilizando *Sigmoid* se alcanzaba un error cuadrático medio de 0.22 luego de 300 iteraciones en los experimentos realizados en la pregunta (f).
- Al utilizar *Cross Validation* el error aumenta a 0.26 para  $K=5$  y  $K=10$ .
- Se puede esperar que para nuevos datos se estime el valor efectivo con un error de 0.26 en lugar de los 0.22 anteriores.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(g)$

- Se utilizó un valor distinto a 0.2 para el *learning rate*
- Con *learning rate* de 0.023 y utilizando *Sigmoid* se alcanzaba un error cuadrático medio de 0.22 luego de 300 iteraciones en los experimentos realizados en la pregunta  $(f)$ .
- Al utilizar *Cross Validation* el error aumenta a 0.26 para  $K=5$  y  $K=10$ .
- Se puede esperar que para nuevos datos se estime el valor efectivo con un error de 0.26 en lugar de los 0.22 anteriores.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(g)$

- Se utilizó un valor distinto a 0.2 para el *learning rate*
- Con *learning rate* de 0.023 y utilizando *Sigmoid* se alcanzaba un error cuadrático medio de 0.22 luego de 300 iteraciones en los experimentos realizados en la pregunta  $(f)$ .
- Al utilizar *Cross Validation* el error aumenta a 0.26 para  $K=5$  y  $K=10$ .
- Se puede esperar que para nuevos datos se estime el valor efectivo con un error de 0.26 en lugar de los 0.22 anteriores.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(g)$

- Se repite el proceso utilizando la función *RELU* como función de activación.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (g)

- Se repite el proceso utilizando la función *ReLU* como función de activación.

```
from sklearn import cross_validation

Xm = X_train_scaled.as_matrix()
ym = y_train_scaled.as_matrix()

kfold = cross_validation.KFold(len(Xm), 5)
cvscores = []
for i, (train, val) in enumerate(kfold):
    # create model
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform', activation='relu'))
    model.add(Dense(1, init='uniform'))
    model.add(Activation('linear'))
    # Compile model
    sgd = SGD(lr=0.2)
    model.compile(optimizer=sgd, loss='mean_squared_error')
    # Fit the model
    model.fit(Xm[train], ym[train], nb_epoch=300, verbose=0)
    # evaluate the model
    scores = model.evaluate(Xm[val], ym[val])
    cvscores.append(scores)

mse_cv = np.mean(cvscores)
print(", MSE con K=5: " + str(mse_cv))
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - (*g*)

- Se repite el proceso utilizando la función *RELU* como función de activación.

```
kfold = cross_validation.KFold(len(Xm), 10)
cvscores = []
for i, (train, val) in enumerate(kfold):
    # create model
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform', activation='relu'))
    model.add(Dense(1, init='uniform'))
    model.add(Activation('linear'))
    # Compile model
    sgd = SGD(lr=0.2)
    model.compile(optimizer=sgd, loss='mean_squared_error')
    # Fit the model
    model.fit(Xm[train], ym[train], nb_epoch=300, verbose=0)
    # evaluate the model
    scores = model.evaluate(Xm[val], ym[val])
    cvscores.append(scores)

mse_cv = np.mean(cvscores)
print(", MSE con K=10:" + str(mse_cv))

32/75 [=====>.....] - ETA: 0s, MSE con K=5: 0.130269209083
32/37 [=====>.....] - ETA: 0s, MSE con K=10: 0.120878713629
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(g)$

- Se utilizó *learning rate* de 0.2 ya que en  $(e)$  nos dio un valor relativamente bajo del error con un valor de 0.02.
- Utilizando *Validación cruzada* el error aumenta a 0.13 para  $K=5$  y 0.13 para  $K=10$ .
- se espera que 0.13 sea una estimación razonable del error cuadrático que se debería observar al utilizar este modelo.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(g)$

- Se utilizó *learning rate* de 0.2 ya que en  $(e)$  nos dio un valor relativamente bajo del error con un valor de 0.02.
- Utilizando *Validación cruzada* el error aumenta a 0.13 para  $K=5$  y 0.13 para  $K=10$ .
- se espera que 0.13 sea una estimación razonable del error cuadrático que se debería observar al utilizar este modelo.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(g)$

- Se utilizó *learning rate* de 0.2 ya que en  $(e)$  nos dio un valor relativamente bajo del error con un valor de 0.02.
- Utilizando *Validación cruzada* el error aumenta a 0.13 para  $K=5$  y 0.13 para  $K=10$ .
- se espera que 0.13 sea una estimación razonable del error cuadrático que se debería observar al utilizar este modelo.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Entrenamiento del modelo utilizando *Progressive Decay*.
- Utilizaremos un *learning rate* del 0.023.
- Utilizando *Progressive Decay* el valor del *learning rate* irá disminuyendo progresivamente tras cada iteración.
- La librería *Keras* utiliza la siguiente función.
- 

$$lr_i = lr_0 \cdot \frac{1}{1 + decay \cdot i}$$

- Donde *decay* es un parámetro de entrada para el método y  $lr_i$  representa el *learning rate* en la iteración  $i$ .



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Entrenamiento del modelo utilizando *Progressive Decay*.
- Utilizaremos un *learning rate* del 0.023.
- Utilizando *Progressive Decay* el valor del *learning rate* irá disminuyendo progresivamente tras cada iteración.
- La librería *Keras* utiliza la siguiente función.
- 

$$lr_i = lr_0 \cdot \frac{1}{1 + decay \cdot i}$$

- Donde *decay* es un parámetro de entrada para el método y  $lr_i$  representa el *learning rate* en la iteración  $i$ .



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Entrenamiento del modelo utilizando *Progressive Decay*.
- Utilizaremos un *learning rate* del 0.023.
- Utilizando *Progressive Decay* el valor del *learning rate* irá disminuyendo progresivamente tras cada iteración.
- La librería *Keras* utiliza la siguiente función.
- 

$$lr_i = lr_0 \cdot \frac{1}{1 + decay \cdot i}$$

- Donde *decay* es un parámetro de entrada para el método y  $lr_i$  representa el *learning rate* en la iteración  $i$ .



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Entrenamiento del modelo utilizando *Progressive Decay*.
- Utilizaremos un *learning rate* del 0.023.
- Utilizando *Progressive Decay* el valor del *learning rate* irá disminuyendo progresivamente tras cada iteración.
- La librería *Keras* utiliza la siguiente función.

•

$$lr_i = lr_0 \cdot \frac{1}{1 + decay \cdot i}$$

- Donde *decay* es un parámetro de entrada para el método y  $lr_i$  representa el *learning rate* en la iteración  $i$ .



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Entrenamiento del modelo utilizando *Progressive Decay*.
- Utilizaremos un *learning rate* del 0.023.
- Utilizando *Progressive Decay* el valor del *learning rate* irá disminuyendo progresivamente tras cada iteración.
- La librería *Keras* utiliza la siguiente función.
- 

$$lr_i = lr_0 \cdot \frac{1}{1 + decay \cdot i}$$

- Donde *decay* es un parámetro de entrada para el método y  $lr_i$  representa el *learning rate* en la iteración  $i$ .



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Entrenamiento del modelo utilizando *Progressive Decay*.
- Utilizaremos un *learning rate* del 0.023.
- Utilizando *Progressive Decay* el valor del *learning rate* irá disminuyendo progresivamente tras cada iteración.
- La librería *Keras* utiliza la siguiente función.
- 

$$lr_i = lr_0 \cdot \frac{1}{1 + decay \cdot i}$$

- Donde *decay* es un parámetro de entrada para el método y  $lr_i$  representa el *learning rate* en la iteración  $i$ .



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Código python:



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Código python:

```
n_decay = 10
lear_decay = np.logspace(-6,0,n_decay)

plt.figure(figsize=(14, 6))

for i in range(n_decay):
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform'))
    model.add(Activation('sigmoid'))
    model.add(Dense(1, init='uniform'))
    model.add(Activation('linear'))
    sgd = SGD(lr=0.023, decay=lear_decay[i])
    model.compile(optimizer=sgd, loss='mean_squared_error')
    hist = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), nb_epoch=300, verbose=0, validation_data=(X_test_scaled.as_matrix(), y_test_scaled.as_matrix()))
    plt.loglog(hist.epoch, hist.history['loss'], label=str(round(lear_decay[i],5)))
    print("Progressive decay: "+str(round(lear_decay[i],6)) + ", Ultimo MSE de Entrenamiento: " + str(round(hist.history['loss'][-1], 3)))

plt.title("MSE vs N de epochas")
plt.xlabel("Epochas")
plt.ylabel("MSE")
plt.grid(True)
leg = plt.legend(title="Decay")
# Ancho de lineas en legend
for legobj in leg.legendHandles:
    legobj.set_linewidth(4.0)
plt.show()
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

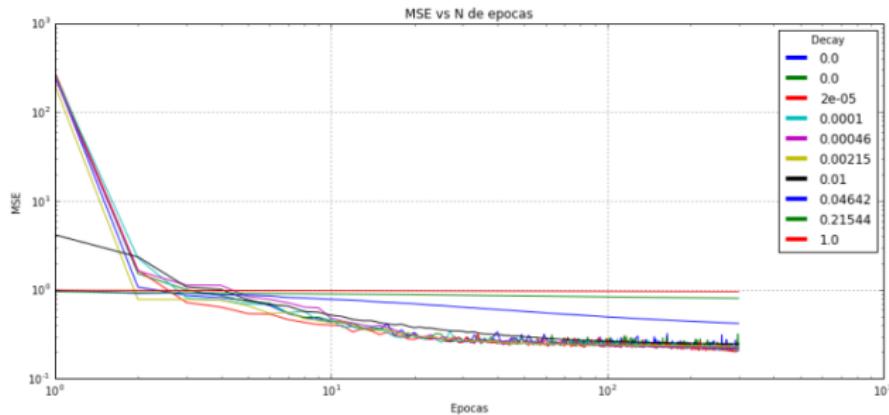
- Gráfico:



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Gráfico:

```
Progressive decay: 1e-06, Ultimo MSE de Entrenamiento: 0.224
Progressive decay: 5e-06, Ultimo MSE de Entrenamiento: 0.251
Progressive decay: 2.2e-05, Ultimo MSE de Entrenamiento: 0.207
Progressive decay: 0.0001, Ultimo MSE de Entrenamiento: 0.22
Progressive decay: 0.000464, Ultimo MSE de Entrenamiento: 0.219
Progressive decay: 0.002154, Ultimo MSE de Entrenamiento: 0.23
Progressive decay: 0.01, Ultimo MSE de Entrenamiento: 0.247
Progressive decay: 0.046416, Ultimo MSE de Entrenamiento: 0.42
Progressive decay: 0.215443, Ultimo MSE de Entrenamiento: 0.803
Progressive decay: 1.0, Ultimo MSE de Entrenamiento: 0.956
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- No cualquier valor del *decay* resulta una mejora al modelo.
- Se observa que los valores del *decay* que llegan al error de 0.22 se encuentran entre  $10^{-6}$  y  $10^{-4}$ .
- Probaremos centrándonos en ese rango.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- No cualquier valor del *decay* resulta una mejora al modelo.
- Se observa que los valores del *decay* que llegan al error de 0.22 se encuentran entre  $10^{-6}$  y  $10^{-4}$ .
- Probaremos centrándonos en ese rango.



# PREDICCIÓN DEL PRECIO DE UNA CASA - ( $h$ )

- No cualquier valor del *decay* resulta una mejora al modelo.
- Se observa que los valores del *decay* que llegan al error de 0.22 se encuentran entre  $10^{-6}$  y  $10^{-4}$ .
- Probaremos centrándonos en ese rango.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Centrándonos en el rango  $10^{-6}$  y  $10^{-4}$ :



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Centrándonos en el rango  $10^{-6}$  y  $10^{-4}$ :

```
n_decay = 10
lear_decay = np.logspace(-6,-4,n_decay)

plt.figure(figsize=(14, 6))

for i in range(n_decay):
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform'))
    model.add(Activation('sigmoid'))
    model.add(Dense(1, init='uniform'))
    model.add(Activation('linear'))
    sgd = SGD(lr=0.023, decay=lear_decay[i])
    model.compile(optimizer=sgd, loss='mean_squared_error')
    hist = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), nb_epoch=300, verbose=0, validation_data=(X_test_scaled.as_matrix(), y_test_scaled.as_matrix()))
    plt.loglog(hist.epoch, hist.history['loss'], label=str(round(lear_decay[i],5)))
    print("Progressive decay: "+str(round(lear_decay[i],6)) + ", Último MSE de Entrenamiento: " +
        str(round(hist.history['loss'][-1], 3)))

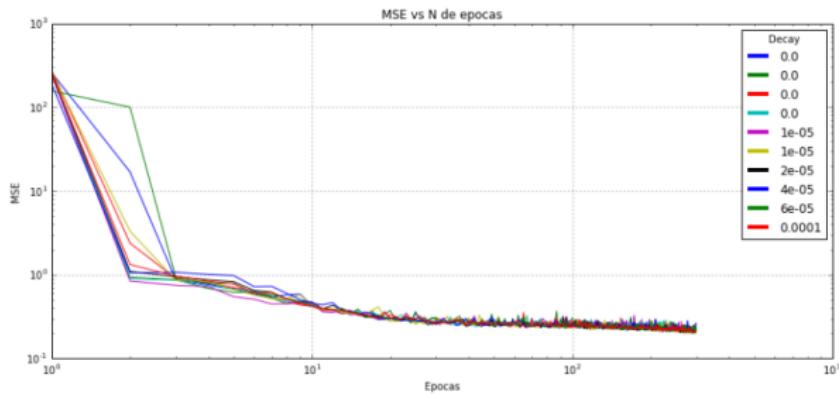
plt.title("MSE vs N de épocas")
plt.xlabel("Epocas")
plt.ylabel("MSE")
plt.grid(True)
leg = plt.legend(title="Decay")
# Ancho de líneas en legend
for legobj in leg.legendHandles:
    legobj.set_linewidth(4.0)
plt.show()
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Centrándonos en el rango  $10^{-6}$  y  $10^{-4}$ :

Progressive decay: 1e-06, Ultimo MSE de Entrenamiento: 0.264  
Progressive decay: 2e-06, Ultimo MSE de Entrenamiento: 0.223  
Progressive decay: 3e-06, Ultimo MSE de Entrenamiento: 0.206  
Progressive decay: 5e-06, Ultimo MSE de Entrenamiento: 0.204  
Progressive decay: 8e-06, Ultimo MSE de Entrenamiento: 0.216  
Progressive decay: 1.3e-05, Ultimo MSE de Entrenamiento: 0.214  
Progressive decay: 2.2e-05, Ultimo MSE de Entrenamiento: 0.214  
Progressive decay: 3.6e-05, Ultimo MSE de Entrenamiento: 0.225  
Progressive decay: 6e-05, Ultimo MSE de Entrenamiento: 0.223  
Progressive decay: 0.0001, Ultimo MSE de Entrenamiento: 0.215



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- En esta ocasión se observa que para configuraciones de *decay* entre  $3 * 10^{-6}$  y  $5 * 10^{-6}$  se logra menor error cuadrático.
- Se repite una vez más el experimento, esta vez sobre una distribución homogénea de valores en escala lineal entre  $10^{-6}$  y  $10^{-5}$ .



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- En esta ocasión se observa que para configuraciones de *decay* entre  $3 * 10^{-6}$  y  $5 * 10^{-6}$  se logra menor error cuadrático.
- Se repite una vez más el experimento, esta vez sobre una distribución homogénea de valores en escala lineal entre  $10^{-6}$  y  $10^{-5}$ .



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Centrándonos en el rango  $10^{-6}$  y  $10^{-5}$ :



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Centrándonos en el rango  $10^{-6}$  y  $10^{-5}$ :

```
n_decay = 10
lear_decay = np.linspace(0.000001,0.00001,n_decay)

plt.figure(figsize=(14, 6))

for i in range(n_decay):
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform'))
    model.add(Activation('sigmoid'))
    model.add(Dense(1, init='uniform'))
    model.add(Activation('linear'))
    sgd = SGD(lr=0.023, decay=lear_decay[i])
    model.compile(optimizer=sgd, loss='mean_squared_error')
    hist = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), nb_epoch=300, verb
ose=0, validation_data=(X_test_scaled.as_matrix(), y_test_scaled.as_matrix()))
    plt.loglog(hist.epoch, hist.history['loss'], label=str(round(lear_decay[i],5)))
    print("Progressive decay: "+str(round(lear_decay[i],6)) + ", Ultimo MSE de Entrenamiento: "
+ str(round(hist.history['loss'][-1], 3)))

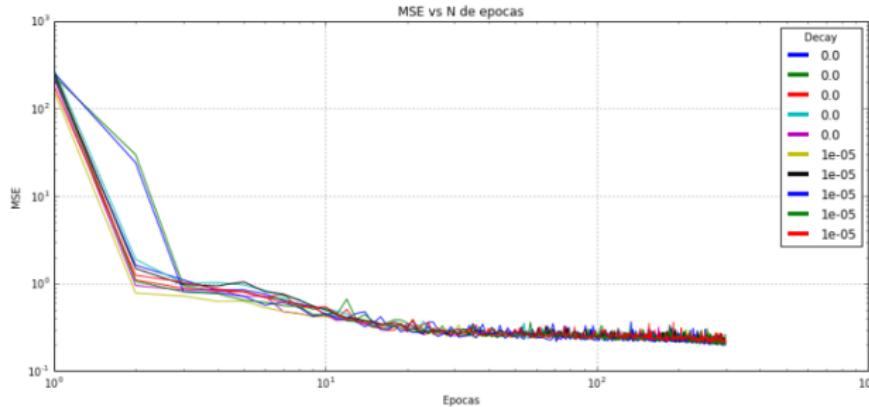
plt.title("MSE vs N de epochas")
plt.xlabel("Epochas")
plt.ylabel("MSE")
plt.grid(True)
leg = plt.legend(title="Decay")
# Ancho de lineas en legend
for legobj in leg.legendHandles:
    legobj.set_linewidth(4.0)
plt.show()
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Centrándonos en el rango  $10^{-6}$  y  $10^{-5}$ :

```
Progressive decay: 1e-06, Ultimo MSE de Entrenamiento: 0.204
Progressive decay: 2e-06, Ultimo MSE de Entrenamiento: 0.205
Progressive decay: 3e-06, Ultimo MSE de Entrenamiento: 0.211
Progressive decay: 4e-06, Ultimo MSE de Entrenamiento: 0.225
Progressive decay: 5e-06, Ultimo MSE de Entrenamiento: 0.2
Progressive decay: 6e-06, Ultimo MSE de Entrenamiento: 0.258
Progressive decay: 7e-06, Ultimo MSE de Entrenamiento: 0.219
Progressive decay: 8e-06, Ultimo MSE de Entrenamiento: 0.198
Progressive decay: 9e-06, Ultimo MSE de Entrenamiento: 0.214
Progressive decay: 1e-05, Ultimo MSE de Entrenamiento: 0.214
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Se alcanzan algunos valores con error más bajo que previo al uso de *progressive decay*.
- Aleatoriedad en el aprendizaje.
- *Progressive decay* con valores no adecuados perjudica el desempeño del modelo.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Se alcanzan algunos valores con error más bajo que previo al uso de *progressive decay*.
- Aleatoriedad en el aprendizaje.
- *Progressive decay* con valores no adecuados perjudica el desempeño del modelo.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(h)$

- Se alcanzan algunos valores con error más bajo que previo al uso de *progressive decay*.
- Aleatoriedad en el aprendizaje.
- *Progressive decay* con valores no adecuados perjudica el desempeño del modelo.





# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Entrenamiento del modelo utilizando *Momentum*.
- Estrategia que persigue mejorar la convergencia durante el entrenamiento.
- Se agrega a lo anterior una componente de memoria
- Se puede interpretar como un factor de inercia.
- Mitiga el riesgo de mínimos locales.
- Mejora la velocidad y error de convergencia.





# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Entrenamiento del modelo utilizando *Momentum*.
- Estrategia que persigue mejorar la convergencia durante el entrenamiento.
- Se agrega a lo anterior una componente de memoria
- Se puede interpretar como un factor de inercia.
- Mitiga el riesgo de mínimos locales.
- Mejora la velocidad y error de convergencia.





# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Entrenamiento del modelo utilizando *Momentum*.
- Estrategia que persigue mejorar la convergencia durante el entrenamiento.
- Se agrega a lo anterior una componente de memoria
- Se puede interpretar como un factor de inercia.
- Mitiga el riesgo de mínimos locales.
- Mejora la velocidad y error de convergencia.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Entrenamiento del modelo utilizando *Momentum*.
- Estrategia que persigue mejorar la convergencia durante el entrenamiento.
- Se agrega a lo anterior una componente de memoria
- Se puede interpretar como un factor de inercia.
- Mitiga el riesgo de mínimos locales.
- Mejora la velocidad y error de convergencia.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Entrenamiento del modelo utilizando *Momentum*.
- Estrategia que persigue mejorar la convergencia durante el entrenamiento.
- Se agrega a lo anterior una componente de memoria
- Se puede interpretar como un factor de inercia.
- Mitiga el riesgo de mínimos locales.
- Mejora la velocidad y error de convergencia.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Entrenamiento del modelo utilizando *Momentum*.
- Estrategia que persigue mejorar la convergencia durante el entrenamiento.
- Se agrega a lo anterior una componente de memoria
- Se puede interpretar como un factor de inercia.
- Mitiga el riesgo de mínimos locales.
- Mejora la velocidad y error de convergencia.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (*i*)

- Utilizando distintas proporciones de *momentum*, función de activación *sigmoid* y *learning rate* de 0.023.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Utilizando distintas proporciones de *momentum*, función de activación *sigmoid* y *learning rate* de 0.023.

```
n_decay = 21
momentum = np.linspace(0,1,n_decay)

plt.figure(figsize=(14, 8))

for i in range(n_decay):
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform', activation='sigmoid'))
    model.add(Dense(1, init='uniform'))
    model.add(Activation('linear'))
    sgd = SGD(lr=0.023,momentum=momentum[i])
    model.compile(optimizer=sgd,loss='mean_squared_error')
    hist = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), nb_epoch=300, verbose=0, validation_data=(X_test_scaled.as_matrix(), y_test_scaled.as_matrix()))
    plt.loglog(hist.epoch, hist.history['loss'], label=str(round(momentum[i], 2)))
    print("Momentum: "+str(round(momentum[i],3)) + ", Ultimo MSE de Entrenamiento: " + str(round(hist.history['loss'][-1], 3)))

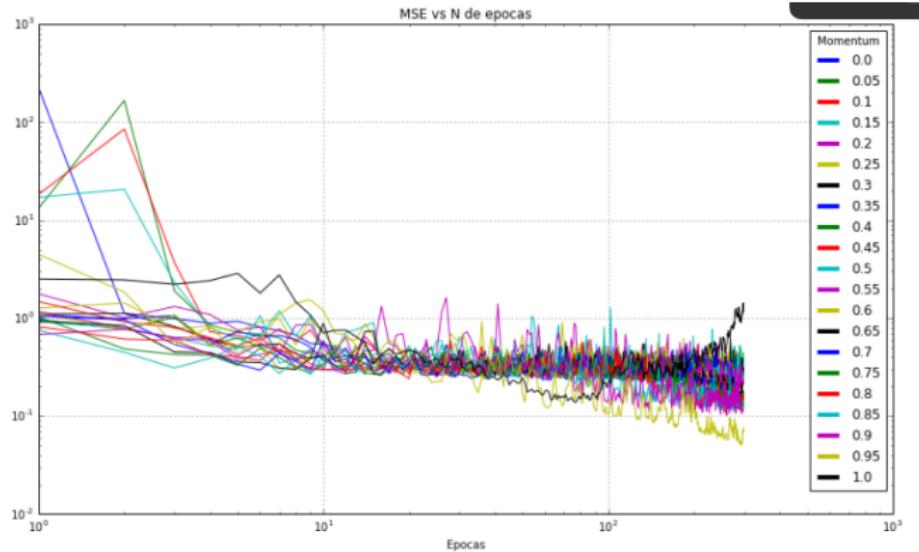
plt.title("MSE vs N de epochas")
plt.xlabel("Epochas")
plt.ylabel("MSE")
plt.grid(True)
leg = plt.legend(title="Momentum")
# Ancho de lineas en legend
for legobj in leg.legendHandles:
    legobj.set_linewidth(4.0)

plt.show()
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Utilizando distintas proporciones de *momentum*, función de activación *sigmoid* y *learning rate* de 0.023.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Con *momentum* 0 o relativamente bajo no se genera error significativo en el error cuadrático de convergencia.
- Al aumentar a valores intermedios también puede aumentar el error.
- Con el valor *momentum* igual a 0.95 se aprecia que el modelo logra valores consistentemente bajos a partir de 100 iteraciones.
- Repetiremos el experimento utilizando variantes en el valor del *momentum* alrededor de esa vecindad.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Con *momentum* 0 o relativamente bajo no se genera error significativo en el error cuadrático de convergencia.
- Al aumentar a valores intermedios también puede aumentar el error.
- Con el valor *momentum* igual a 0.95 se aprecia que el modelo logra valores consistentemente bajos a partir de 100 iteraciones.
- Repetiremos el experimento utilizando variantes en el valor del *momentum* alrededor de esa vecindad.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Con *momentum* 0 o relativamente bajo no se genera error significativo en el error cuadrático de convergencia.
- Al aumentar a valores intermedios también puede aumentar el error.
- Con el valor *momentum* igual a 0.95 se aprecia que el modelo logra valores consistentemente bajos a partir de 100 iteraciones.
- Repetiremos el experimento utilizando variantes en el valor del *momentum* alrededor de esa vecindad.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Con *momentum* 0 o relativamente bajo no se genera error significativo en el error cuadrático de convergencia.
- Al aumentar a valores intermedios también puede aumentar el error.
- Con el valor *momentum* igual a 0.95 se aprecia que el modelo logra valores consistentemente bajos a partir de 100 iteraciones.
- Repetiremos el experimento utilizando variantes en el valor del *momentum* alrededor de esa vecindad.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (*i*)

- Variando los valores de *momentum* alrededor de 0.95.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Variando los valores de *momentum* alrededor de 0.95.

```
n_decay = 21
momentum = np.linspace(0.9,1,n_decay)

plt.figure(figsize=(14, 8))

for i in range(n_decay):
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform', activation='sigmoid'))
    model.add(Dense(1, init='uniform'))
    model.add(Activation('linear'))
    sgd = SGD(lr=0.023,momentum=momentum[i])
    model.compile(optimizer=sgd, loss='mean_squared_error')
    hist = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), nb_epoch=300, verbose=0, validation_data=(X_test_scaled.as_matrix(), y_test_scaled.as_matrix()))
    plt.loglog(hist.epoch, hist.history['loss'], label=str(round(momentum[i], 2)))
    print("Momentum: "+str(round(momentum[i],3)) + ", Ultimo MSE de Entrenamiento: " + str(round(hist.history['loss'])[-1], 3))

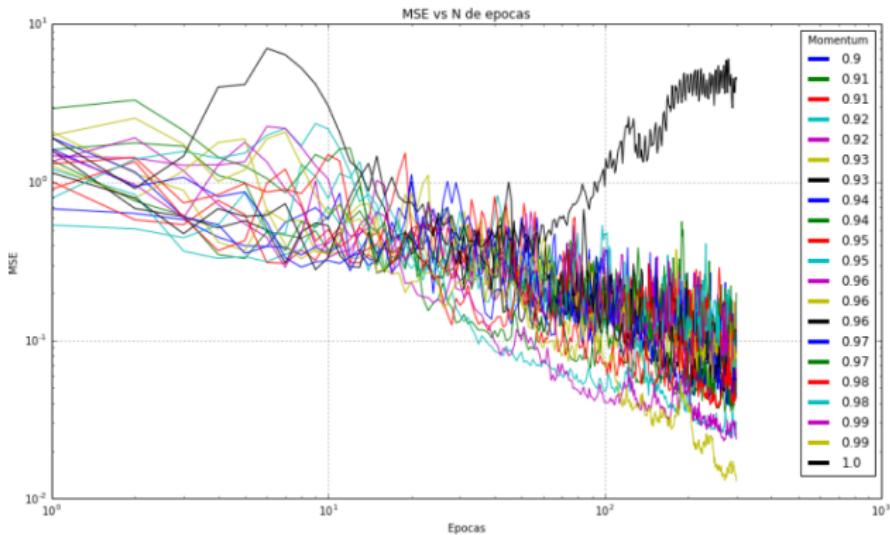
plt.title("MSE vs N de épocas")
plt.xlabel("Epocas")
plt.ylabel("MSE")
plt.grid(True)
leg = plt.legend(title="Momentum")
# Ancho de líneas en legend
for legobj in leg.legendHandles:
    legobj.set_linewidth(4.0)

plt.show()
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Variando los valores de *momentum* alrededor de 0.95.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Con *momentum* 0 o relativamente bajo no se genera error significativo en el error cuadrático de convergencia.
- Al aumentar a valores intermedios también puede aumentar el error.
- Con el valor *momentum* igual a 0.95 se aprecia que el modelo logra valores consistentemente bajos a partir de 100 iteraciones.
- Repetiremos el experimento utilizando variantes en el valor del *momentum* alrededor de esa vecindad.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Con *momentum* 0 o relativamente bajo no se genera error significativo en el error cuadrático de convergencia.
- Al aumentar a valores intermedios también puede aumentar el error.
- Con el valor *momentum* igual a 0.95 se aprecia que el modelo logra valores consistentemente bajos a partir de 100 iteraciones.
- Repetiremos el experimento utilizando variantes en el valor del *momentum* alrededor de esa vecindad.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Con *momentum* 0 o relativamente bajo no se genera error significativo en el error cuadrático de convergencia.
- Al aumentar a valores intermedios también puede aumentar el error.
- Con el valor *momentum* igual a 0.95 se aprecia que el modelo logra valores consistentemente bajos a partir de 100 iteraciones.
- Repetiremos el experimento utilizando variantes en el valor del *momentum* alrededor de esa vecindad.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Con *momentum* 0 o relativamente bajo no se genera error significativo en el error cuadrático de convergencia.
- Al aumentar a valores intermedios también puede aumentar el error.
- Con el valor *momentum* igual a 0.95 se aprecia que el modelo logra valores consistentemente bajos a partir de 100 iteraciones.
- Repetiremos el experimento utilizando variantes en el valor del *momentum* alrededor de esa vecindad.



# PREDICCIÓN DEL PRECIO DE UNA CASA - (*i*)

- Se aprecia que el *momentum* que más logra reducir el error final es justo antes de alcanzar el valor 1.
- Utilizar una alta proporción de *momentum*, mientras se mantenga una mínima componente de gradiente, ha resultado muy ventajoso para encontrar el óptimo global.
- Probable *Overfitting*



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Se aprecia que el *momentum* que más logra reducir el error final es justo antes de alcanzar el valor 1.
- Utilizar una alta proporción de *momentum*, mientras se mantenga una mínima componente de gradiente, ha resultado muy ventajoso para encontrar el óptimo global.
- Probable *Overfitting*



# PREDICCIÓN DEL PRECIO DE UNA CASA - (i)

- Se aprecia que el *momentum* que más logra reducir el error final es justo antes de alcanzar el valor 1.
- Utilizar una alta proporción de *momentum*, mientras se mantenga una mínima componente de gradiente, ha resultado muy ventajoso para encontrar el óptimo global.
- Probable *Overfitting*



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Entrenamiento del modelo utilizando *Batch*.
- *Batch gradient descent*.
- *Online gradient descent*.
- *Mini Batch*.
- Se aplicará a continuación distintas configuraciones de *batches* al modelo construido con *sigmoid* y *learning rate* de 0.023..



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Entrenamiento del modelo utilizando *Batch*.
- *Batch gradient descent*.
- *Online gradient descent*.
- *Mini Batch*.
- Se aplicará a continuación distintas configuraciones de *batches* al modelo construido con *sigmoid* y *learning rate* de 0.023..



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Entrenamiento del modelo utilizando *Batch*.
- *Batch gradient descent*.
- *Online gradient descent*.
- *Mini Batch*.
- Se aplicará a continuación distintas configuraciones de *batches* al modelo construido con *sigmoid* y *learning rate* de 0.023..



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Entrenamiento del modelo utilizando *Batch*.
- *Batch gradient descent*.
- *Online gradient descent*.
- *Mini Batch*.
- Se aplicará a continuación distintas configuraciones de *batches* al modelo construido con *sigmoid* y *learning rate* de 0.023..



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Entrenamiento del modelo utilizando *Batch*.
- *Batch gradient descent*.
- *Online gradient descent*.
- *Mini Batch*.
- Se aplicará a continuación distintas configuraciones de *batches* al modelo construido con *sigmoid* y *learning rate* de 0.023..



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Configuraciones de *batches* usando *sigmoid* y *learning rate* de 0.023.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Configuraciones de *batches* usando *sigmoid* y *learning rate* de 0.023.

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

n_batches = 21

batch_sizes = np.round(np.linspace(1,X_train_scaled.shape[0],n_batches))
hist = []

for b in batch_sizes:
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform', activation='sigmoid'))
    model.add(Dense(1, init='uniform'))
    model.add(Activation('linear'))
    sgd = SGD(lr=0.023)
    model.compile(optimizer=sgd, loss='mean_squared_error')
    hist3 = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), batch_size=b, nb_epoch=300, verbose=0)
    print("Tamaño de Batch: " + str(b) + ", Último MSE de Entrenamiento: " + str(round(hist3.history['loss'][[-1], 3])))
    hist.append(model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), batch_size=b, nb_epoch=300, verbose=0))

lhist = np.zeros_like(batch_sizes)

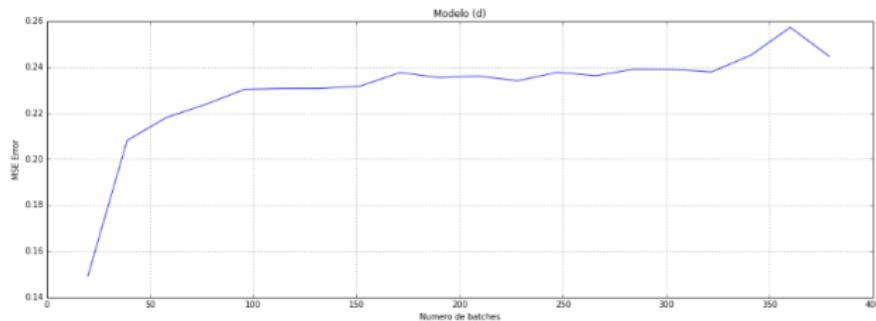
for i in range(n_batches):
    lhist[i] = np.mean(hist[i].history['loss']) # promedio de errores de cada batch

plt.figure(figsize=(18, 6))
plt.plot(batch_sizes, lhist)
plt.grid(True)
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Configuraciones de *batches* usando *sigmoid* y *learning rate* de 0.023.





# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Se observa en primer lugar que aparentemente la red entrenada con *sigmoid* no es apropiada para entrenamiento online.
- A continuación se observa que con *batches* desde 20 unidades ya se obtiene convergencia, y con error cuadrático significativamente menores que con *bath gradient descent*. Se estima entonces que la implementación de *mini-batch gradient descent* ha resultado ventajosa en este caso.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Se observa en primer lugar que aparentemente la red entrenada con *sigmoid* no es apropiada para entrenamiento online.
- A continuación se observa que con *batches* desde 20 unidades ya se obtiene convergencia, y con error cuadrático significativamente menores que con *bath gradient descent*. Se estima entonces que la implementación de *mini-batch gradient descent* ha resultado ventajosa en este caso.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Utilizando ahora la función *Relu*.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Utilizando ahora la función *Relu*.

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

n_batches = 21

batch_sizes = np.round(np.linspace(1,X_train_scaled.shape[0],n_batches))
hist = []

for b in batch_sizes:
    model = Sequential()
    model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform', activation='relu'))
    model.add(Dense(1, init='uniform'))
    model.add(Activation('linear'))
    sgd = SGD(lr=0.023)
    model.compile(optimizer=sgd, loss='mean_squared_error')
    hist3 = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), batch_size=b, nb_epoch=300, verbose=0)
    print("Tamaño de Batch: " + str(b) + ", Último MSE de Entrenamiento: " + str(round(hist3.history['loss'][-1], 3)))
    hist.append(model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), batch_size=b, nb_epoch=300, verbose=0))

lhist = np.zeros_like(batch_sizes)

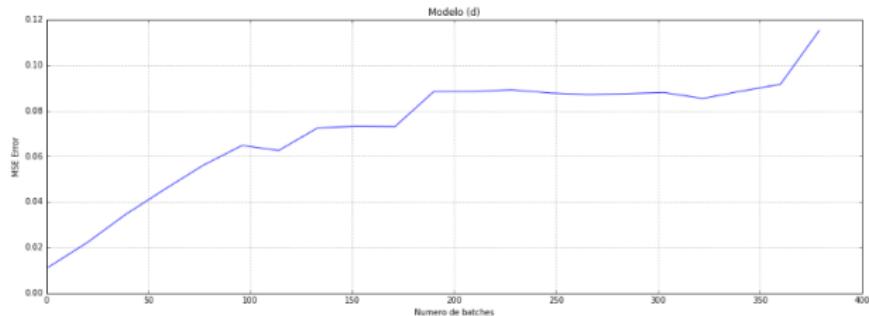
for i in range(n_batches):
    lhist[i] = np.mean(hist[i].history['loss']) # promedio de errores de cada batch

plt.figure(figsize=(18, 6))
plt.plot(batch_sizes, lhist)
plt.grid(True)
plt.title("Modelo (d)")
plt.xlabel("Número de batches")
plt.ylabel("MSE Error")
plt.show()
```



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Utilizando ahora la función *Relu*.



# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Con *ReLU* sí se obtiene convergencia para *online gradient descent*.
- Se aprecia también que en la medida que se aplica *mini-batch gradient descent* con tamaño progresivamente más grande, consistentemente aumenta el error.

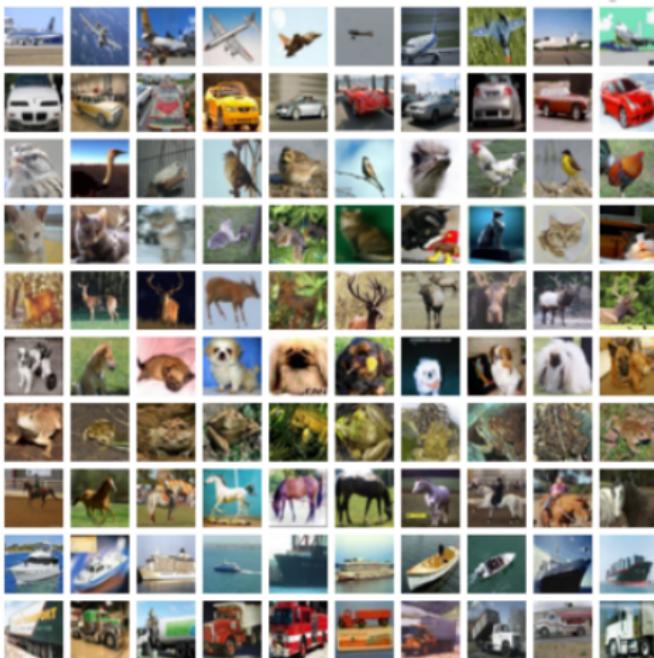


# PREDICCIÓN DEL PRECIO DE UNA CASA - $(j)$

- Con *ReLU* sí se obtiene convergencia para *online gradient descent*.
- Se aprecia también que en la medida que se aplica *mini-batch gradient descent* con tamaño progresivamente más grande, consistentemente aumenta el error.



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10





# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (A)

- Función que carga todos los bloques de entrenamiento y pruebas:



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (A)

- Función que carga todos los bloques de entrenamiento y pruebas:

```
import cPickle as pickle
import numpy as np
import os
from scipy.misc import imread

def load_CIFAR_one(filename):
    with open(filename, 'rb') as f:
        datadict = pickle.load(f)
    X = datadict['data']
    Y = datadict['labels']
    Y = np.array(Y)
    return X, Y

def load_CIFAR10(PATH):
    xs = []
    ys = []
    for b in range(1,6):
        f = os.path.join(PATH, 'data_batch_%d' % (b, ))
        X, Y = load_CIFAR_one(f)
        xs.append(X)
        ys.append(Y)
    Xtr = np.concatenate(xs)
    Ytr = np.concatenate(ys)
    del X, Y
    Xte, Yte = load_CIFAR_one(os.path.join(PATH, 'test_batch'))
    return Xtr, Ytr, Xte, Yte
```



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (B)

- Función que escala apropiadamente las imágenes antes de trabajar:



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (B)

- Función que escala apropiadamente las imágenes antes de trabajar:

```
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")

def preprocess(centering, scaling):
    Xtr, Ytr, Xte, Yte = load_CIFAR10("datasets/")
    scalerX = StandardScaler(with_mean=centering, with_std=scaling).fit(Xtr)

    # Normalización del conjunto de entrenamiento
    Xtr_scaled = scalerX.transform(Xtr)

    # Normalización del conjunto de testing
    Xte_scaled = scalerX.transform(Xte)

    del Xtr, Xte
    return Xtr_scaled, Ytr, Xte_scaled, Yte
```





# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (B)

- Utilizando *Scikit Learn* podemos centrar y escalar los datos.
- Para centrar sólo restamos la media de cada columna.
- Para centrar y escalar los datos, además de restar la media, cada columna es dividida por la desviación estándar.





# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (B)

- Utilizando *Scikit Learn* podemos centrar y escalar los datos.
- Para centrar sólo restamos la media de cada columna.
- Para centrar y escalar los datos, además de restar la media, cada columna es dividida por la desviación estándar.



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (B)

- Utilizando *Scikit Learn* podemos centrar y escalar los datos.
- Para centrar sólo restamos la media de cada columna.
- Para centrar y escalar los datos, además de restar la media, cada columna es dividida por la desviación estándar.



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (C)

- Diseñe, entrene y evalúe una red neuronal con salida softmax para el problema CIFAR a partir de la representación original de las imágenes (píxeles RGB).



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (C)

```
%matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np

def show_sample(Xtr, Ytr, label_name):
    r, g, b = np.split(Xtr, 3)
    rgbArray = np.zeros((32,32,3), 'uint8')
    rgbArray[:, :, 0] = np.reshape(r, (-1, 32))
    rgbArray[:, :, 1] = np.reshape(g, (-1, 32))
    rgbArray[:, :, 2] = np.reshape(b, (-1, 32))
    img = Image.fromarray(rgbArray)
    print(label_name)
    plt.figure(figsize=(1,1))
    plt.axis('off')
    plt.imshow(img)
    plt.show()

label_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship',
               'truck']

Xtr, Ytr, _, _ = load_CIFAR10("datasets/")

for i in range(5):
    show_sample(Xtr[i], Ytr[i], label_names[Ytr[i]])
```



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (c)

frog



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (c)

truck



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (c)

truck



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (c)

deer



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (c)

automobile



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (c)

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD
from keras.utils import np_utils

Xtr, Ytr, Xte, Yte = preprocess(True, True)

nb_classes = 10

# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(Ytr, nb_classes)
Y_test = np_utils.to_categorical(Yte, nb_classes)

model = Sequential()

# Input layer
model.add(Dense(64, input_dim=Xtr.shape[1], init='uniform'))
model.add(Activation('tanh'))
model.add(Dropout(0.5))

# Hidden layer
model.add(Dense(64, init='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.5))

# Hidden layer
model.add(Dense(64, init='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.5))
```



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (c)

```
# Hidden layer
model.add(Dense(64, init='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.5))

# Hidden layer
model.add(Dense(64, init='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.5))

# Hidden layer
model.add(Dense(64, init='uniform'))
model.add(Activation('relu'))
model.add(Dropout(0.5))

# Output layer
model.add(Dense(10, init='uniform'))
model.add(Activation('softmax'))

sgd = SGD(lr=0.2, decay=1e-6, momentum=0.9)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

hist = model.fit(Xtr, Y_train, nb_epoch=400, batch_size=3072, verbose=0)
score = model.evaluate(Xte, Y_test, batch_size=3072)
print(score)

10000/10000 [=====] - 1s
[1.5930610433578491, 0.43560000538825988]
```



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (C)

Los modelos experimentados fueron los siguientes:

• **Modelo 1:**

- Capa de entrada: 3072 neuronas.
- Capa oculta 1: 128 neuronas. Función de activación: Tangente Hiperbólica.
- Capa oculta 2: 128 neuronas. Función de activación: Lineal.
- Capa de salida: 10 neuronas. Función de activación: Softmax.
- Método de entrenamiento: SGD:
  - Learning rate: 0.2
  - Decay: 1e-6
  - Momentum: 0.9
- Función de costo: Categorical Crossentropy
- Número de épocas: 400.
- Tamaño de batch: 3072.
- **Ajuste del modelo: 47%**

• **Modelo 2:**

- Capa de entrada: 3072 neuronas.
- Capa oculta 1: 64 neuronas. Función de activación: ReLu.
- Capa oculta 2: 64 neuronas. Función de activación: ReLu.
- Capa de salida: 10 neuronas. Función de activación: Softmax.
- Método de entrenamiento: SGD:
  - Learning rate: 0.01
  - Decay: 1e-6
  - Momentum: 0.9
- Función de costo: Categorical Crossentropy
- Número de épocas: 200



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (C)

- **Modelo 3:**

- Capa de entrada: 3072 neuronas.
- Capa oculta 1: 64 neuronas. Función de activación: Tangente Hiperbólica.
- Capa oculta 2: 64 neuronas. Función de activación: Tangente Hiperbólica.
- Capa de salida: 10 neuronas. Función de activación: Softmax.
- Método de entrenamiento: SGD:
  - Learning rate: 0.01
  - Decay: 1e-6
  - Momentum: 0.9
- Función de costo: Categorical Crossentropy
- Número de épocas: 200.
- Tamaño de batch: 32.
- **Ajuste del modelo: 21%**

- **Modelo 4:**

- Capa de entrada: 3072 neuronas.
- Capa oculta 1: 64 neuronas. Función de activación: Tangente Hiperbólica.
- Capa oculta 2: 64 neuronas. Función de activación: Tangente Hiperbólica.
- Capa de salida: 10 neuronas. Función de activación: Softmax.
- Método de entrenamiento: SGD:
  - Learning rate: 0.01
  - Decay: 1e-6
  - Momentum: 0.9
- Función de costo: Categorical Crossentropy
- Número de épocas: 200.



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (C)

- **Modelo 5:**

- Capa de entrada: 3072 neuronas.
- Capa oculta 1: 64 neuronas. Función de activación: Tangente Hiperbólica.
- Capa oculta 2: 64 neuronas. Función de activación: ReLu.
- Capa oculta 3: 64 neuronas. Función de activación: ReLu.
- Capa oculta 4: 64 neuronas. Función de activación: ReLu.
- Capa de salida: 10 neuronas. Función de activación: Softmax.
- Método de entrenamiento: SGD:
  - Learning rate: 0.2
  - Decay: 1e-6
  - Momentum: 0.9
- Función de costo: Categorical Crossentropy
- Número de épocas: 400.
- Tamaño de batch: 3072.
- **Ajuste del modelo: 44%**



# RECONOCIMIENTO DE IMÁGENES EN CIFAR10 - (D)

- Diseñe, entrene y evalúe una red neuronal con salida softmax para el problema CIFAR a partir de la representación original de las imágenes (píxeles RGB).



# OUTLINE

1 INTRODUCCIÓN

2 OBJETIVOS

3 DESARROLLO

4 CONCLUSIONES

5 REFERENCIAS



# CONCLUSIONES

Gracias al desarrollo de este trabajo logramos:

- Adquirir conocimiento práctico respecto de la implementación mediante *Keras* de modelos de redes neuronales artificiales.
- Diferenciar los dos tipos de problemas más comunes en *Machine Learning*: Problemas de *Regresión* y *Clasificación*.
- Introducir a la noción de *Paradigmas* de las ANN. Número de capas, neuronas, funciones de activación, etc...



# OUTLINE

1 INTRODUCCIÓN

2 OBJETIVOS

3 DESARROLLO

4 CONCLUSIONES

5 REFERENCIAS



## REFERENCIAS

-  Hastie, T.; Tibshirani, R., Friedman, J. (2009), The Elements of Statistical Learning, Second Edition. Springer New York Inc.
-  Bishop, Christopher M. (1995). Neural Networks for Pattern Recognition, Clarendon Press.
-  Krizhevsky, A., Hinton, G. (2009). Learning multiple layers of features from tiny images.



# REFERENCIAS

-  Harrison, D. and Rubinfeld, D. (1978). Hedonic prices and the demand for clean air, *Journal of Environmental Economics and Management*, 5, 81-102
-  Dalal, N., Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* (Vol. 1, pp.886-893). IEEE.
-  Forsyth, D. A., Ponce, J. (2002). *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference.

