Artificial (deep) Neural Networks

Prediction of house value: the California housing dataset

Statistical Learning – Part II

Alberto Castellini
University of Verona

Overview

- Keras: Python deep learning library
- Case study and dataset: the California housing dataset
- Neural network model generation
- Exercise

Libraries

Keras

Reference: https://keras.io/

Keras is:
- a **high-level neural networks API**
- written in **Python**
- capable of running on top of **TensorFlow** and other libraries
- developed with a focus on **enabling fast experimentation**

Use Keras if you need a **deep learning library that:**
- allows for **easy and fast prototyping**
- supports both **convolutional networks** and **recurrent networks**
- runs seamlessly on **CPU** and **GPU**

Keras is **compatible with Python 2.7-3.6**

TensorFlow and other libraries/environments

TensorFlow

TensorFlow is:

- an **end-to-end open source platform** for **machine learning**
- **comprehensive**, **flexible ecosystem of tools**, **libraries** and community resources
- A tool for **easily build** and **deploy ML powered applications**
- Reference: https://www.tensorflow.org/

PyTorch

PyTorch is an open source machine learning framework that accelerates the path from research prototyping to production deployment.

Reference: https://pytorch.org/

colab

Colab is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.

Reference: https://colab.research.google.com/

Case study and dataset

Dataset: California Housing

This dataset was derived from the 1990 U.S. census, using one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).

References:

- on Scikit-learn: https://scikit-learn.org/stable/datasets/index.html#california-housing-dataset
- on StatLib repository: http://lib.stat.cmu.edu/datasets/
- Kaggle datasets: https://www.kaggle.com/camnugent/california-housing-prices

Dataset: California Housing

Dataset characteristics:

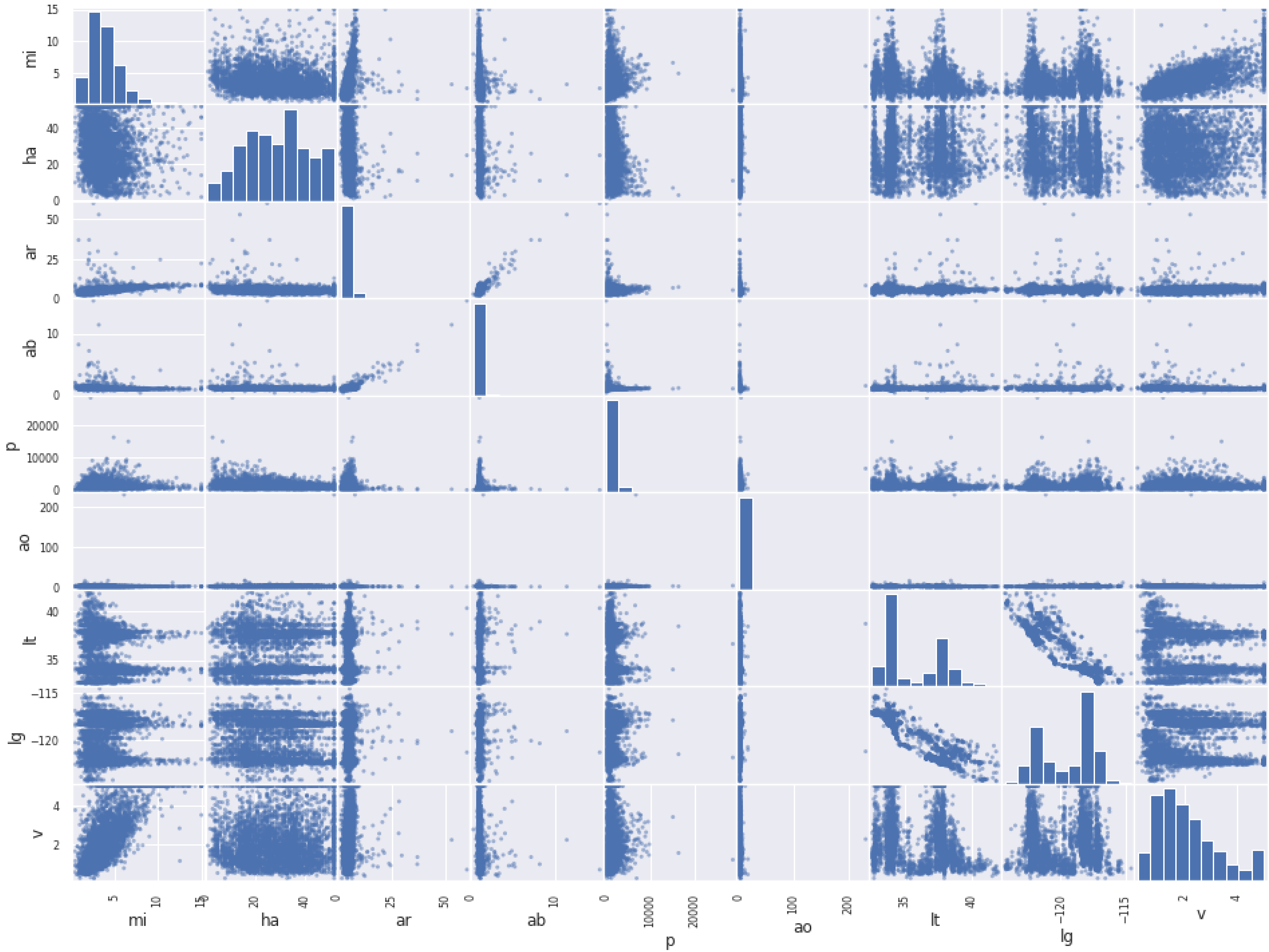
- # instances: **20640**

- # variables: **8** numeric predictors, **1** target

- Variable names:
    - ***MedInc (mi)****:* median income in block
    - ***HouseAge (ha)****:* median house age in block
    - ***AveRooms (ar)****:* average number of rooms
    - ***AveBedrms (ab)****:* average number of bedrooms
    - ***Population (p)****:* block population
    - ***AveOccup (ao)****:* average house occupancy
    - ***Latitude (lt)****:* house block latitude
    - ***Longitude (lg)****:* house block longitude
    - ***Target (v):*** median house value for California districts

- Missing values: none

Scatter matrix



Artificial Neural Networks - model generation

Model creation and usage with Keras: a toy example

model = **Sequential**()   **# 1**

model.**add**(Dense(10, input_dim=X_train.shape[1], activation='relu')) **# 2**

model.**add**(Dense(30, activation='relu')) **# 3**

model.**add**(Dense(40, activation='relu'))

model.**add**(Dense(1)) **# 4**

model.**compile**(optimizer ='adam', loss = 'mean_squared_error', metrics =[metrics.mae]) **# 5**

history = model.**fit**(X_train, y_train, validation_data=(X_val, y_val), epochs=150, batch_size=32) **# 6**

model.**summary**() **# 7**

pred = model.**predict**(X_test) **# 8**

1. Generation of *Sequential* model

model = **Sequential**()   **# 1**

- The *Sequential* model is a **linear stack of layers**

- It can be created
  i) by passing a list of layer instances to the constructor
  ii) by adding layers after the creation via the *.add()* method

Reference:

- Getting started with the Keras Sequential model (https://keras.io/getting-started/sequential-model-guide/)

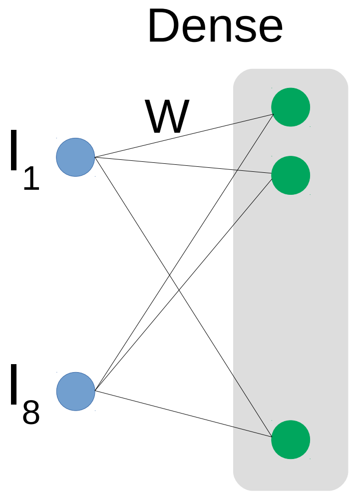
2. Addition of input layer

model.**add**(**Dense**(**10**, **input_dim**=X_train.shape[1], **activation**='relu')) **# 2**

- The model needs to know what **input shape** it should expect
- The first layer in a *Sequential* model (and **only the first**, because following layers can do **automatic shape inference**) needs to receive information about its **input shape**
- **Dense**: implements the operation:

$$output = activation(dot(input, kernel) + bias)$$

  - *activation* is the element-wise activation function
  - *kernel* is a weights matrix
  - *bias* is a bias vector

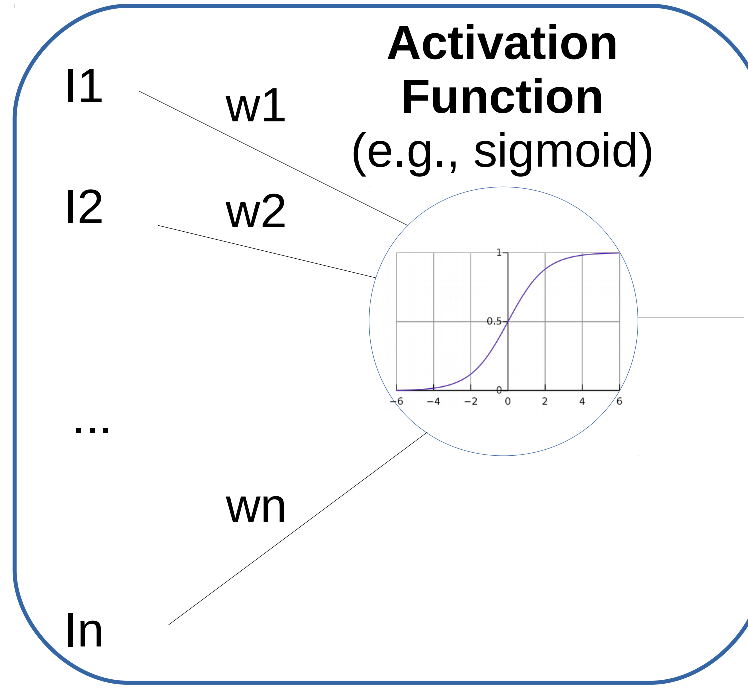


Reference:
- Keras documentation: https://keras.io/layers/core/

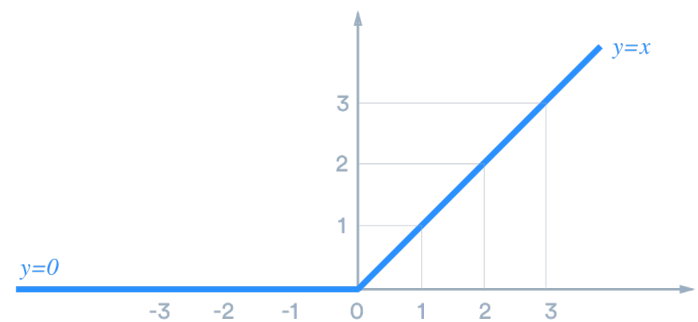
2a. Activation functions

Available activation functions:

- *sigmoid*
- *hard sigmoid*
- *softmax*
- *tanh*: hyperbolic tangent activation function
- *relu*: rectified linear unit



$$f(x) = \text{element-wise max}(x,0)$$
$$\text{where } x = \text{sum}_j(I_j * w_j + b_j)$$



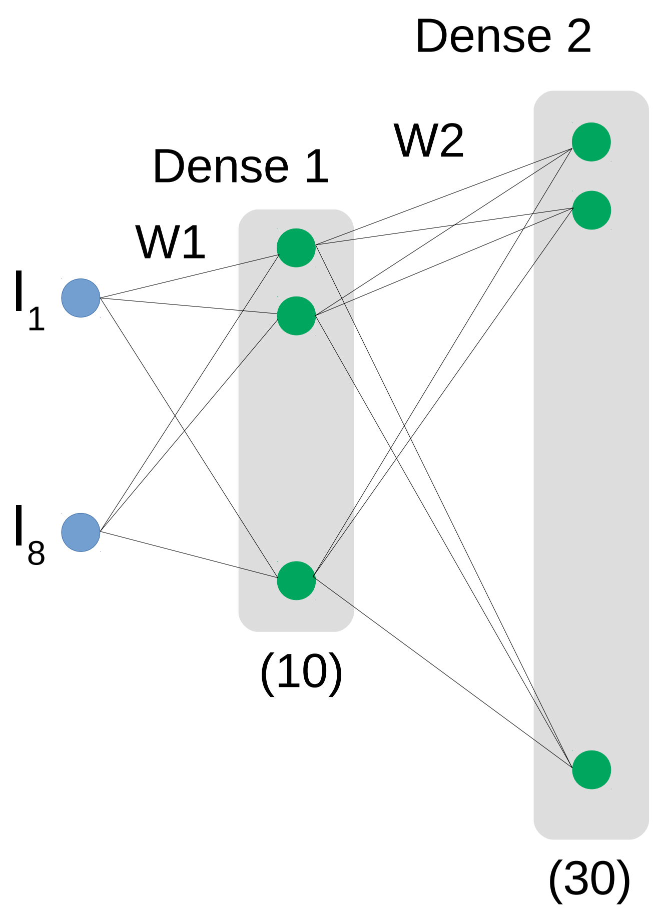
Reference:

- Keras documentation: https://keras.io/activations/

3. Addition of internal layer

```
model.add(Dense(30, activation='relu'))  # 3
```

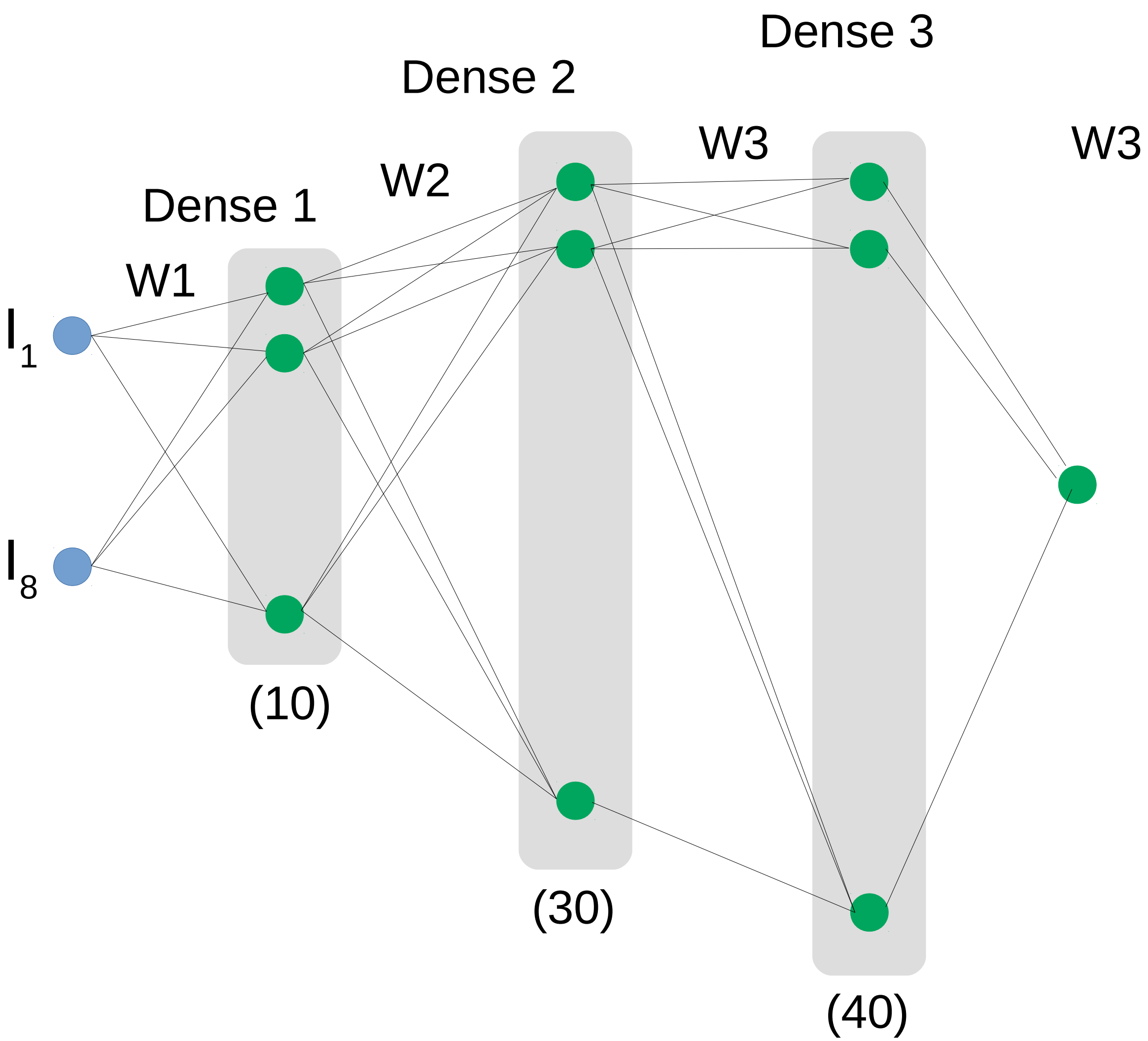
- Following layers can do **automatic shape inference**



4. Output layer

model.add(Dense(1)) **# 4**

- Following layers can do **automatic shape inference**



5. Compilation

model.**compile**(**optimizer** ='adam', **loss** = 'mean_squared_error', **metrics** =[metrics.mae]) **# 5**

- Before training a model, you need to configure the learning process via the *compile* method

- Input:
  - **Optimizer** (e.g., adam, see https://arxiv.org/abs/1412.6980v8): an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments
    Ref: https://keras.io/optimizers/

  - **Loss function**: the objective that the model will try to minimize
    Ref: https://keras.io/losses/

  - **A list of metrics**: used to judge the performance of your model (e.g., accuracy, mean absolute error)
    Ref: https://keras.io/metrics/

6. Training

history = model.**fit**(X_train, y_train, **validation_data**=(X_val, y_val),
epochs=150, **batch_size**=32) **# 6**

- Keras models are trained on **Numpy arrays** of input data and labels

  # Load data
  df_train = pd.read_csv("../input/test.csv", index_col=0)
  df_test = pd.read_csv("../input/train.csv", index_col=0)

  Data load

  df_train_np = df_train.values
  df_test_np = df_test.values

  Conversion to numpy array

- **validation_data**: data on which to evaluate the loss and any model metrics at the end of each epoch

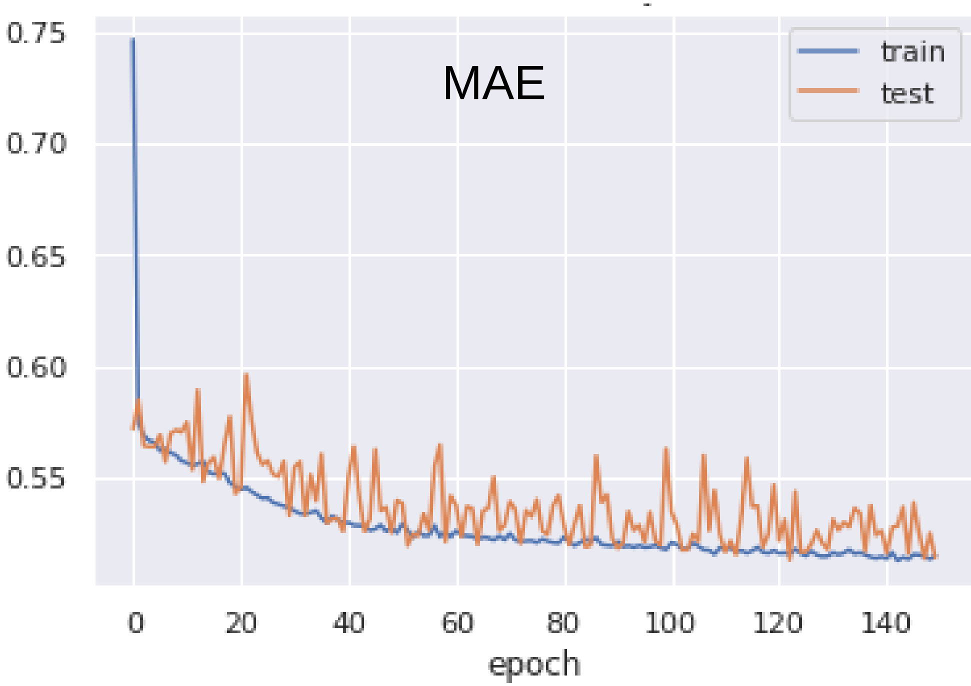
- **epochs**: number of iterations of the training phase

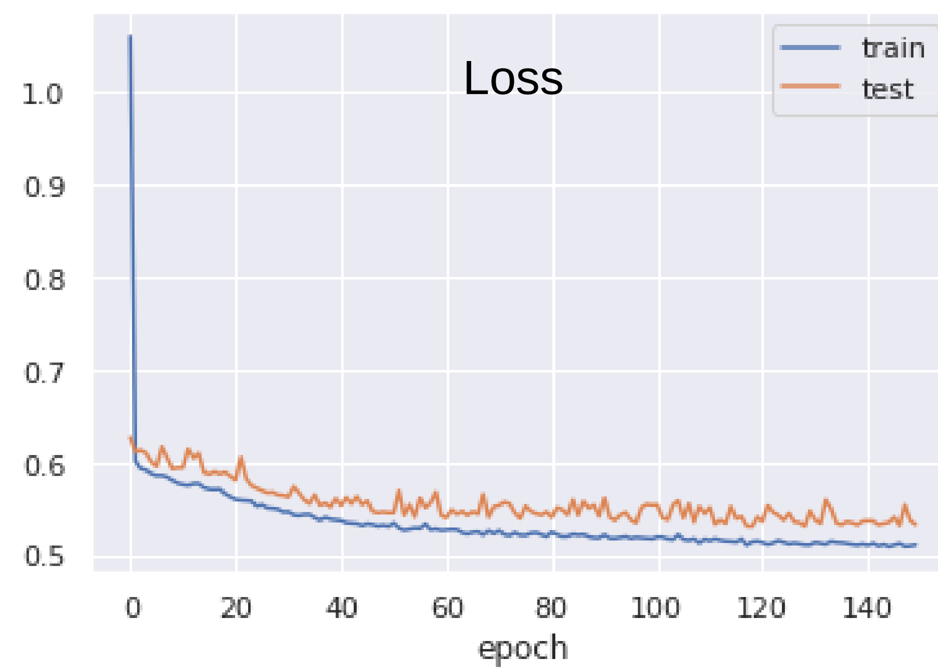
- **batch_size**: number of samples per gradient update (default: 32)

6. Training

```
Train on 11469 samples, validate on 4916 samples
Epoch 1/150
11469/11469 [==============================] - 1s 69us/step - loss: 1.0604 - mean_absolute_error: 0.7466 - val_loss: 0.6279 - val_mean_absolute_error: 0.5720
Epoch 2/150
11469/11469 [==============================] - 1s 55us/step - loss: 0.6026 - mean_absolute_error: 0.5747 - val_loss: 0.6126 - val_mean_absolute_error: 0.5851
Epoch 3/150
11469/11469 [==============================] - 1s 54us/step - loss: 0.5946 - mean_absolute_error: 0.5690 - val_loss: 0.6143 - val_mean_absolute_error: 0.5643
Epoch 4/150
11469/11469 [==============================] - 1s 53us/step - loss: 0.5929 - mean_absolute_error: 0.5668 - val_loss: 0.6111 - val_mean_absolute_error: 0.5640
Epoch 5/150
11469/11469 [==============================] - 1s 54us/step - loss: 0.5891 - mean_absolute_error: 0.5659 - val_loss: 0.6018 - val_mean_absolute_error: 0.5640
Epoch 6/150
```

Loss (training) | MAE (training) | Loss (validation) | MAE (validation)





7. Model summary

model.**summary**() **# 7**

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                80
_________________________________________________________________
dense_2 (Dense)              (None, 30)                330
_________________________________________________________________
dense_3 (Dense)              (None, 40)                1240
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 41
=================================================================
Total params: 1,691
Trainable params: 1,691
Non-trainable params: 0
_________________________________________________________________
```

8 x 10

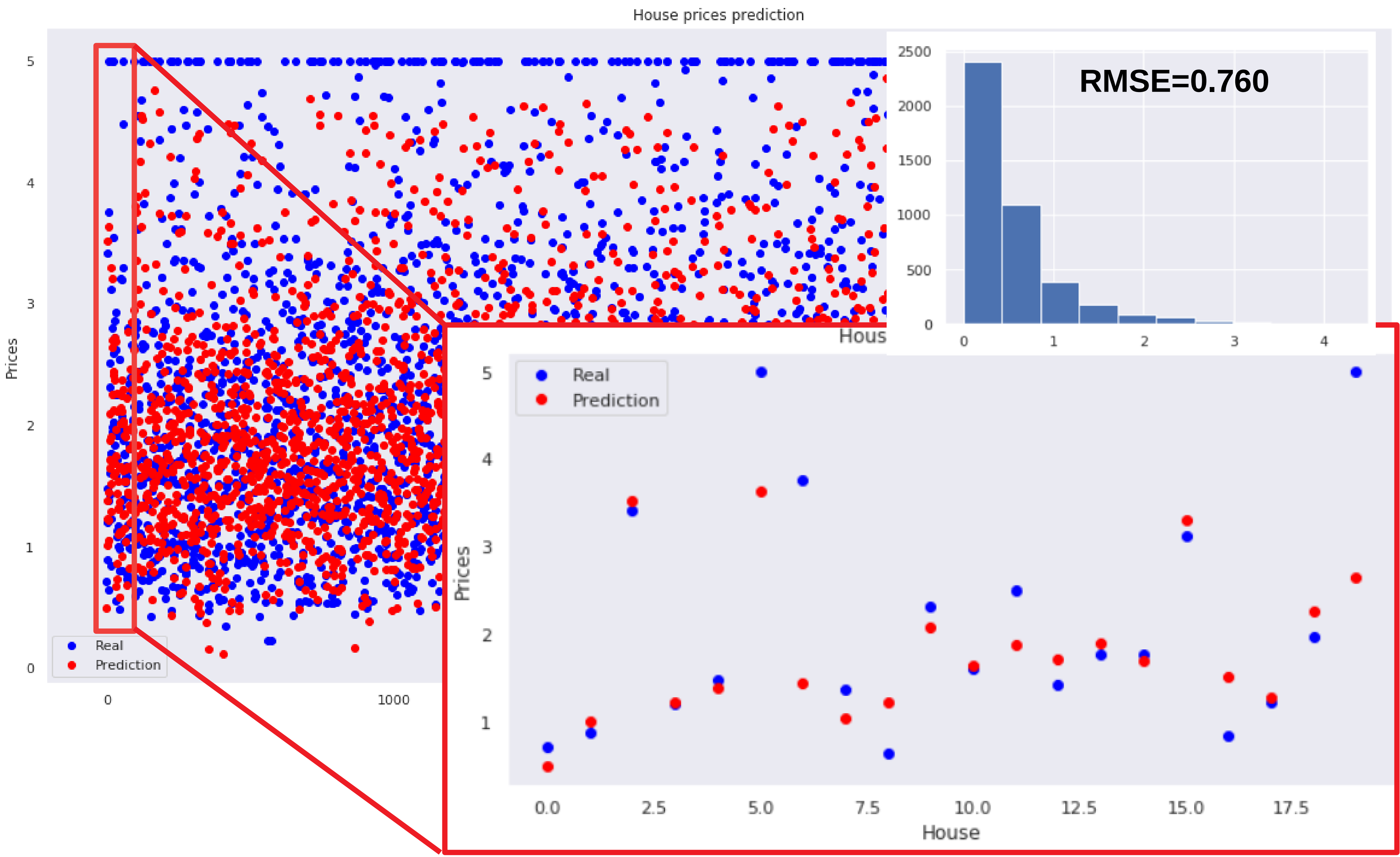
11 x 30

31 x 40

41

8. Prediction (on new data)

pred = model.**predict**(X_test) **# 8**



Exercise

Exercise

- Browse the Keras library (tutorial and documentation cited in the slides)
- Load the California housing dataset
- Generate the artificial neural network model analyzed in this slides and compare the results
- Test the following network structures and compare the results in terms of training/validation MAE/loss, RMSE on test set:
  - 1 layer containing a single neuron
  - 1 layer containing 3 neurons
  - 1 layer containing 10 neurons
  - 2 layers containing respectively 10 and 30 neurons
  - 3 layers containing respectively 10, 30 and 40 neurons
- Generate a chart in which the performance of these models are displayed and compared