Lecture_8a_Neural_Networks

August 15, 2020

1 Lesson 8a

This notebook contains all the sample code and solutions to the exercises in chapter 10.

2 Setup

First, let's make sure this notebook works well in both python 2 and 3, import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures:

[34]:
```
# Python 3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn 0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

try:
    # %tensorflow_version only exists in Colab.
    %tensorflow_version 2.x
except Exception:
    pass

# TensorFlow 2.0 is required
import tensorflow as tf
assert tf.__version__ >= "2.0"

# Common imports
import numpy as np
import os
import pandas as pd
# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "ann"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

# Ignore useless warnings (see SciPy issue #5998)
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")
```

Real neural networks inspired artificial neural networks (ANNs). Though planes were inspired by birds, they don't have to flap their wings. Similarly, ANNs have gradually become quite different from their biological cousins.

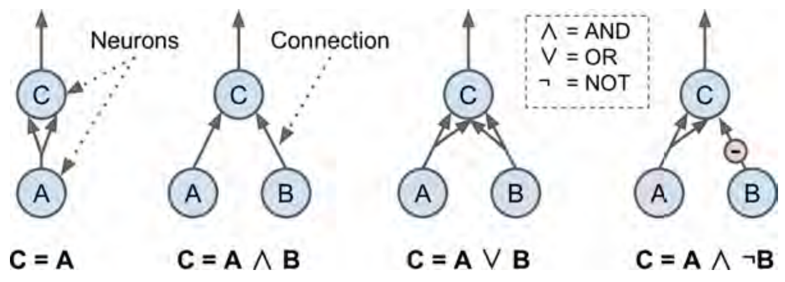
ANNs are at the very core of Deep Learning. They are versatile, powerful, and scalable, making them ideal to tackle large and highly complex Machine Learning tasks, such as classifying billions of images (e.g., Google Images), powering speech recognition services (e.g., Apple's Siri), recommending the best videos to watch to hundreds of millions of users every day (e.g., YouTube), or learning to beat the world champion at the game of Go by examining millions of past games and then playing against itself (DeepMind's AlphaGo).

In this lesson, we will introduce artificial neural networks, starting with a quick tour of the very first ANN architectures. Then we will present Multi-Layer Perceptrons (MLPs) and implement one using TensorFlow to tackle the MNIST digit classification problem.

Surprisingly, ANNs have been around for quite a while: they were first introduced back in 1943 by the neurophysiologist Warren McCulloch and the mathematician Walter Pitts.

From 1960s-1980s - dark age period with little development. In 1990s SVM proposed as better alternative to ANN. From 2010 resurgent of ANN because: * Huge quantity of data. ANN outperform other ML algorithms on large and complex problems * Increase of computation power and thanks to gaming industry powerful GPU that can process ANN very efficiently * Small tweaks of training algorithms that had strong impact * Theoretical limitations of ANN (local optima problem) are rare in practice * Now best brains work on ANN

Warren McCulloch and Walter Pitts proposed a very simple model of the biological neuron, which later became known as an artificial neuron: it has one or more binary (on/off) inputs and one binary output. The artificial neuron simply activates its output when more than a certain number of its inputs are active. For example, let's build a few ANNs that perform various logical computations, assuming that a neuron is activated when at least two of its inputs are active.



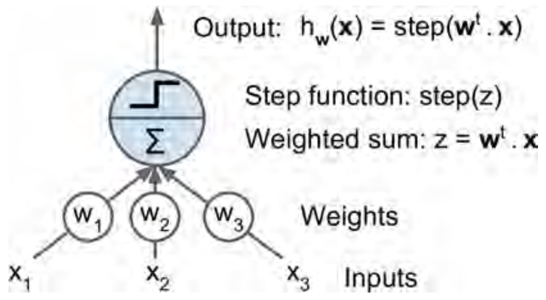
Neurons: 1. Identity function: if neuron A is activated, then neuron C gets activated as well. 2. Neuron C is activated only when **both** neurons A and B are activated 3. Neuron C gets activated if either neuron A **or** neuron B is activated (or both) 4. Neuron C is activated only if neuron A is **active** and if neuron B is **off**.

3 Perceptrons

The Perceptron is one of the simplest ANN architectures, invented in 1957 by Frank Rosenblatt. It is based on artificial neuron called a linear threshold unit (LTU): the inputs and output are now numbers (instead of binary on/off values) and each input connection is associated with a weight.

The LTU computes a weighted sum of its inputs ($z = w_1x_1 + w_2x_2 + ... + w_nx_n = w^T \cdot x$), then applies a step function to that sum and outputs the result:

$$h_w(x) = step(z) = step(w^T \cdot x)$$



Output: $h_w(\mathbf{x}) = step(\mathbf{w}^t . \mathbf{x})$
Step function: step(z)
Weighted sum: $z = \mathbf{w}^t . \mathbf{x}$
Σ
w_1
w_2
w_3
Weights
x_1
x_2
x_3
Inputs

The most common step function used in Perceptrons is the Heaviside step function (discrete function). Sometimes the sign function is used instead:

$$heavyslide(z) = \begin{cases} 0 & if \quad z < 0 \\ 1 & if \quad z \geq 0 \end{cases}$$

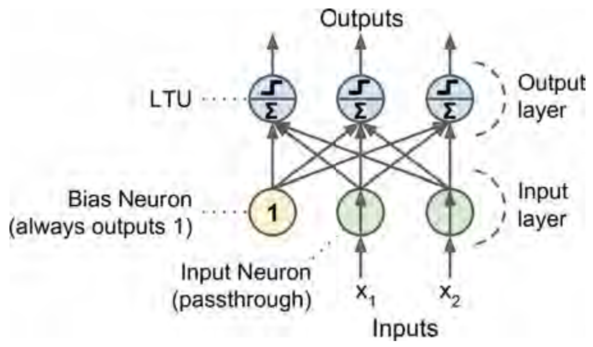
$$sgn(z) = \begin{cases} -1 & if \quad z < 0 \\ 0 & if \quad z = 0 \\ 1 & if \quad z \geq 0 \end{cases}$$

A single LTU can be used for simple linear binary classification. It computes a linear combination of the inputs and if the result exceeds a threshold, it outputs the positive class or else outputs the negative class (just like a Logistic Regression classifier or a linear SVM).

Next we use a single LTU to classify iris flowers based on the petal length and width (also adding an extra bias feature $x_0 = 1$, just like we did in previous chapters). Training an LTU means finding the right values for w_0, w_1, and w_2.

A Perceptron has a single layer of LTUs, with each neuron connected to all the inputs. These connections are often represented using special pass-through neurons called input neurons: they just output whatever input they are fed. Moreover, an extra bias feature is generally added (x0 = 1). This bias feature is typically represented using a special type of neuron called a bias neuron, which just outputs 1 all the time.

A Perceptron with two inputs and three outputs is represented in next. It can classify instances simultaneously into three different binary classes, which makes it a multioutput classifier.



4 Training:

Perceptrons are trained by taking into account the error made by the network; it does not reinforce connections that lead to the wrong output. Perceptron is fed one training instance at a time, and for each instance it makes its predictions.

For every output neuron that produced a wrong prediction, it reinforces the connection weights from the inputs that would have contributed to the correct prediction.

$$w_{i,j}^{next} = w_{i,j} + \eta(\hat{y}_j - y_j)x_i$$

Weights are changes with every instance.

The decision boundary of each output neuron is linear, so Perceptrons are incapable of learning complex patterns. However, if the training instances are linearly separable, the algorithm would converge to a solution.

Scikit-Learn provides a Perceptron class that implements a single LTU network

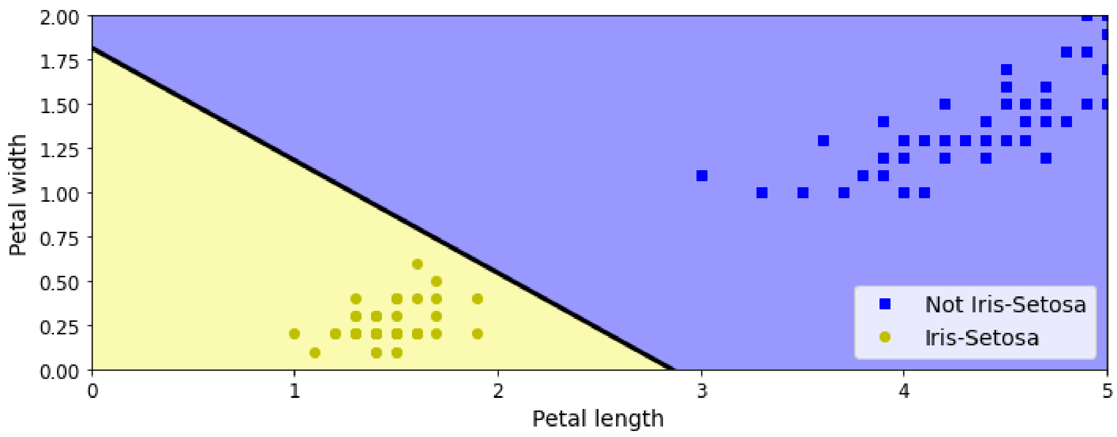
[35]:
```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
```

```
X = iris.data[:, (2, 3)]  # petal length, petal width
y = (iris.target == 0).astype(np.int)

per_clf = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
per_clf.fit(X, y)

y_pred = per_clf.predict([[2, 0.5]])
```

Single Perceptron can only manage one straight line, though it is sufficient in this case.

[36]:
```
a = -per_clf.coef_[0][0] / per_clf.coef_[0][1]
b = -per_clf.intercept_ / per_clf.coef_[0][1]

axes = [0, 5, 0, 2]

x0, x1 = np.meshgrid(
        np.linspace(axes[0], axes[1], 500).reshape(-1, 1),
        np.linspace(axes[2], axes[3], 200).reshape(-1, 1),
    )
X_new = np.c_[x0.ravel(), x1.ravel()]
y_predict = per_clf.predict(X_new)
zz = y_predict.reshape(x0.shape)

plt.figure(figsize=(10, 4))
plt.plot(X[y==0, 0], X[y==0, 1], "bs", label="Not Iris-Setosa")
plt.plot(X[y==1, 0], X[y==1, 1], "yo", label="Iris-Setosa")

plt.plot([axes[0], axes[1]], [a * axes[0] + b, a * axes[1] + b], "k-",␣
 ↪linewidth=3)
from matplotlib.colors import ListedColormap
custom_cmap = ListedColormap(['#9898ff', '#fafab0'])

plt.contourf(x0, x1, zz, cmap=custom_cmap)
plt.xlabel("Petal length", fontsize=14)
plt.ylabel("Petal width", fontsize=14)
plt.legend(loc="lower right", fontsize=14)
plt.axis(axes)

save_fig("perceptron_iris_plot")
plt.show()
```

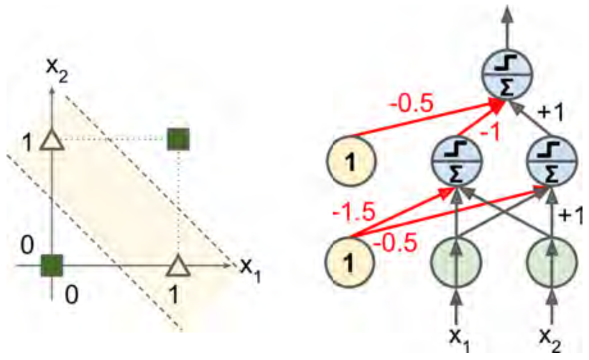
```
Saving figure perceptron_iris_plot
```



Perceptron learning algorithm strongly resembles Stochastic Gradient Descent, Scikit-Learn's Perceptron class is equivalent to using an SGDClassifier with the following hyperparameters: loss="perceptron", learning_rate="constant", eta0=1 (the learning rate), and penalty=None (no regularization).

Perceptrons has number of serious weaknesses of Perceptrons, in particular the fact that they are incapable of solving some trivial problems (e.g., the Exclusive OR (XOR) classification problem: An XOr function should return a true value if the two inputs are not equal and a false value if they are equal.

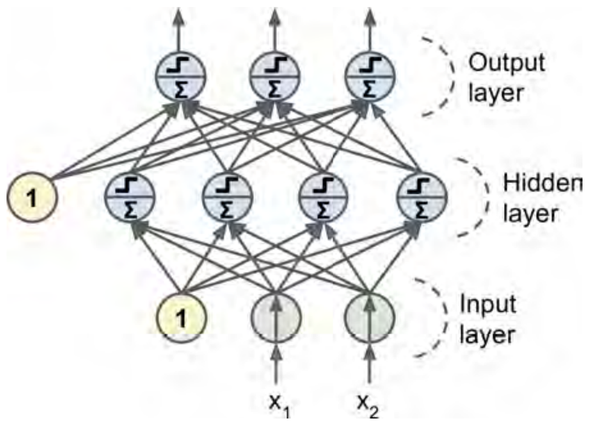
Some of the limitations of Perceptrons can be eliminated by stacking multiple Perceptrons. The resulting ANN is called a Multi-Layer Perceptron (MLP). In particular, an MLP can solve the XOR problem, as you can verify by computing the output of the MLP represented next, for each combination of inputs: with inputs $(0, 0)$ or $(1, 1)$ the network outputs 0, and with inputs $(0, 1)$ or $(1, 0)$ it outputs 1.



- left-bottom: sends negative number in all cases except $x_1 = x_2 = 1$
- right-bottom: sends positive number in all cases except $x_1 = x_2 = 0$
- if left-bottom is negative or right-bottom is positive – send output to 1

5 Multi-Layer Perceptron and Backpropagation

An MLP is composed of one (passthrough) input layer, one or more layers of LTUs, called hidden layers, and one final layer of LTUs called the output layer. Every layer except the output layer includes a bias neuron and is fully connected to the next layer. When an ANN has two or more hidden layers, it is called a deep neural network (DNN).



The MLP is solved using backpropagation training algorithm that uses Gradient Descent using reverse-mode autodiff.

1. For each training instance, the algorithm feeds it to the network and computes the output of every neuron in each consecutive layer (this is the forward pass, just like when making predictions).
2. Then it measures the network's output error (i.e., the difference between the desired output and the actual output of the network), and it computes how much each neuron in the last hidden layer contributed to each output neuron's error.
3. It then proceeds to measure how much of these error contributions came from each neuron in the previous hidden layer and so on until the algorithm reaches the input layer.
4. This reverse pass efficiently measures the error gradient across all the connection weights in the network by propagating the error gradient backward in the network (hence the name of the algorithm).
5. Forward and reverse passes of backpropagation simply perform reverse-mode autodiff. For each node we measure the gradient of y $\frac{\partial y}{\partial n_i}$ and the gradient of MSE y $\frac{\partial MSE}{\partial n_i}$.
6. The last step of the backpropagation algorithm is a Gradient Descent step on all the connection weights in the network, using the error gradients measured earlier.

The key change to the MLP's architecture is the replacement of the step function with the logistic function, $\sigma(z) = 1/(1 + \exp(-z))$. This was essential because the step function contains only flat segments, so there is no gradient to work with (Gradient Descent cannot move on a flat surface), while the logistic function has a well-defined nonzero derivative everywhere, allowing Gradient Descent to make some progress at every step.

The backpropagation algorithm may be used with other activation functions, instead of the logistic function. Two other popular activation functions are:

- The hyperbolic tangent function $\tanh(z) = 2\sigma(2z) - 1$. Just like the logistic function it is S-shaped, continuous, and differentiable, but its output value ranges from –1 to 1 (instead of

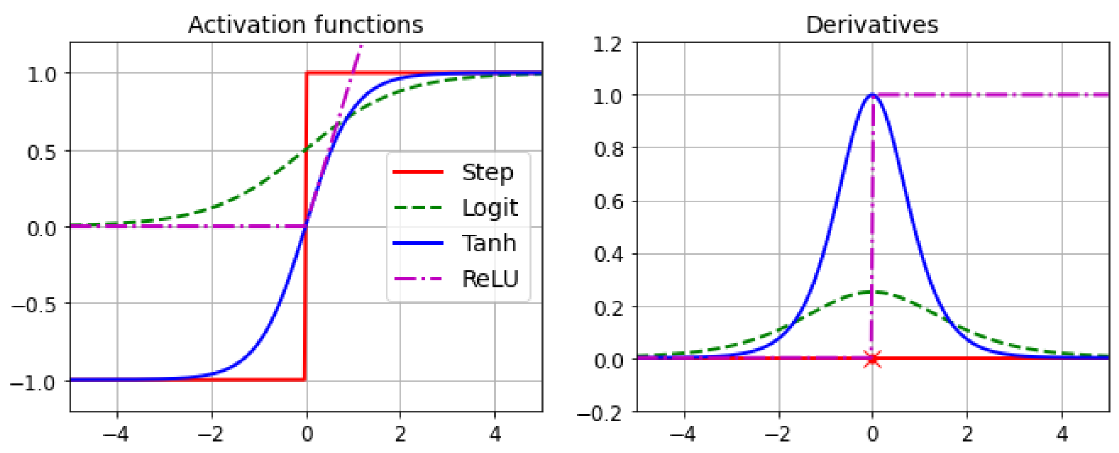
0 to 1 in the case of the logistic function), which tends to make each layer's output centered around 0 at the beginning of training. This often helps speed up convergence.

- The ReLU function: $ReLU(z) = \max(0, z)$. It is continuous but not differentiable at z = 0. However, in practice it works very well and is very fast to compute. Most importantly, the fact that it does not have a maximum output value also helps reduce some issues during Gradient Descent, which we will discuss in the next lesson.

6 Activation functions

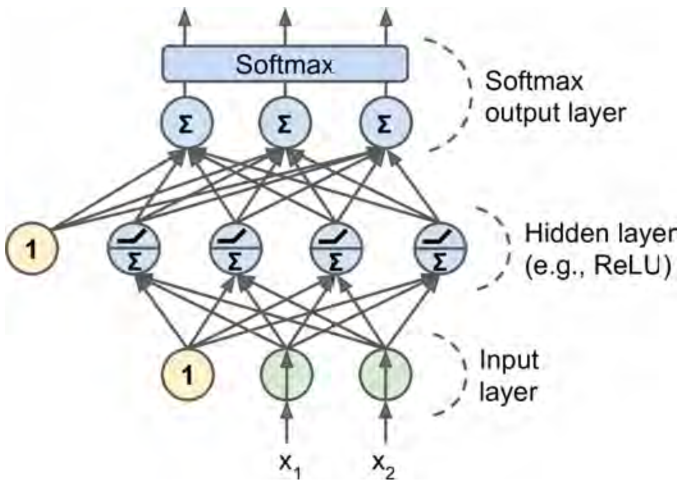
[37]:
```
def logit(z):
    return 1 / (1 + np.exp(-z))

def relu(z):
    return np.maximum(0, z)

def derivative(f, z, eps=0.000001):
    return (f(z + eps) - f(z - eps))/(2 * eps)
```

[38]:
```
z = np.linspace(-5, 5, 200)

plt.figure(figsize=(11,4))

plt.subplot(121)
plt.plot(z, np.sign(z), "r-", linewidth=2, label="Step")
plt.plot(z, logit(z), "g--", linewidth=2, label="Logit")
plt.plot(z, np.tanh(z), "b-", linewidth=2, label="Tanh")
plt.plot(z, relu(z), "m-.", linewidth=2, label="ReLU")
plt.grid(True)
plt.legend(loc="center right", fontsize=14)
plt.title("Activation functions", fontsize=14)
plt.axis([-5, 5, -1.2, 1.2])

plt.subplot(122)
plt.plot(z, derivative(np.sign, z), "r-", linewidth=2, label="Step")
plt.plot(0, 0, "ro", markersize=5)
plt.plot(0, 0, "rx", markersize=10)
plt.plot(z, derivative(logit, z), "g--", linewidth=2, label="Logit")
plt.plot(z, derivative(np.tanh, z), "b-", linewidth=2, label="Tanh")
plt.plot(z, derivative(relu, z), "m-.", linewidth=2, label="ReLU")
plt.grid(True)
#plt.legend(loc="center right", fontsize=14)
plt.title("Derivatives", fontsize=14)
plt.axis([-5, 5, -0.2, 1.2])

plt.show()
```



An MLP is often used for classification, with each output corresponding to a different binary class (e.g., spam/ham, urgent/not-urgent, and so on). When the classes are exclusive (e.g., classes 0 through 9 for digit image classification), the output layer is typically modified by replacing the individual activation functions by a shared soft-max (multinomial logit) function (see figure). The output of each neuron corresponds to the estimated probability of the corresponding class.

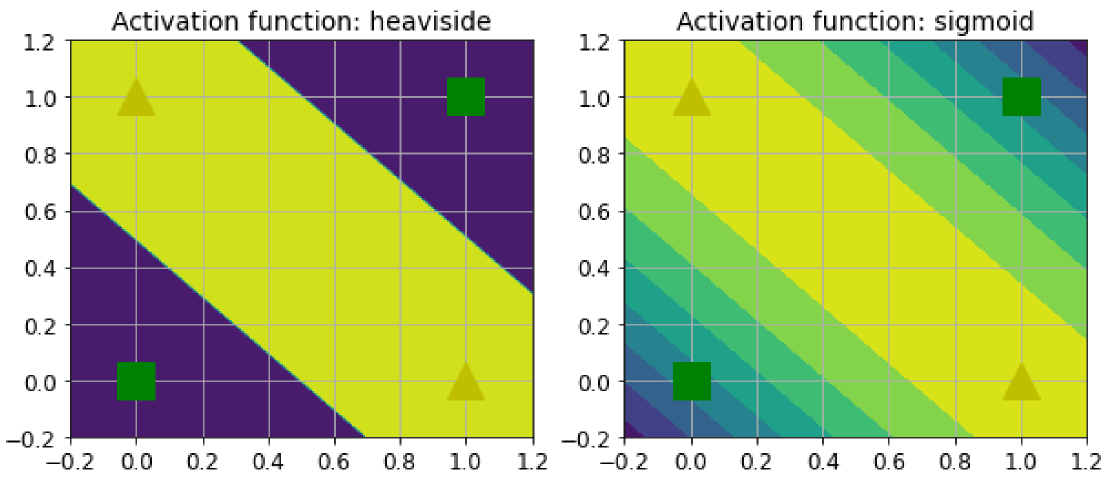
The signal flows only in one direction (from the inputs to the outputs), so this architecture is an example of a feedforward neural network (FNN). Later we will study recurrent networks where the information flows in both directions.



```
[39]: def heaviside(z):
          return (z >= 0).astype(z.dtype)

      def sigmoid(z):
          return 1/(1+np.exp(-z))

      def mlp_xor(x1, x2, activation=heaviside):
```

```
    return activation(-activation(x1 + x2 - 1.5) + activation(x1 + x2 - 0.5) -␣
↪0.5)
```

[40]:
```
x1s = np.linspace(-0.2, 1.2, 100)
x2s = np.linspace(-0.2, 1.2, 100)
x1, x2 = np.meshgrid(x1s, x2s)

z1 = mlp_xor(x1, x2, activation=heaviside)
z2 = mlp_xor(x1, x2, activation=sigmoid)

plt.figure(figsize=(10,4))

plt.subplot(121)
plt.contourf(x1, x2, z1)
plt.plot([0, 1], [0, 1], "gs", markersize=20)
plt.plot([0, 1], [1, 0], "y^", markersize=20)
plt.title("Activation function: heaviside", fontsize=14)
plt.grid(True)

plt.subplot(122)
plt.contourf(x1, x2, z2)
plt.plot([0, 1], [0, 1], "gs", markersize=20)
plt.plot([0, 1], [1, 0], "y^", markersize=20)
plt.title("Activation function: sigmoid", fontsize=14)
plt.grid(True)
```



[40]:
```
```

#KERAS Keras is a high-level Deep Learning API that allows to build, train, evaluate and execture neural networks. References are available in http://www.keras.io and in https://github.com/keras-team/keras. It offers 3 popular Deep Learning libraries: - TensorFlow (Google) - Microsoft Cognitive Toolkit (CNTK) - Theano (University of Montreal)

We will use two implementations of Keras:

Keras supports Tensorflow as a backend. It has a lot of features, more user friendly, but slower than pure Tensorflow.

Tensorflow is mostly designed to software engineers. Very powerful, a bit more complex. Thankfully, Tensorflow offers a Keras, which provides a simpler syntax and high efficiency. We will use both in this class. [Figure 10-10]

Another very populat deep learning library is PyTorch 1.0 (Facebook). The commands are fairly similar to Keras, so it's easy to learn PyTorch once you know Keras.

Any of these tools will work fine most applications. The drawbacks in the previous versions (TensorFlow 1.0, PyTorch prior to 1.0 were addressed).

7 Building an Image Classifier

First let's import TensorFlow and Keras.

[41]:
```
import tensorflow as tf
from tensorflow import keras
```

[42]:
```
tf.__version__
```

[42]:
```
'2.3.0'
```

[43]:
```
keras.__version__
```

[43]:
```
'2.4.0'
```

Let's start by loading the fashion MNIST dataset. Keras has a number of functions to load popular datasets in `keras.datasets`. The fashion items are more difficult to classify than numbers. The dataset is already split for you between a training set and a test set, but it can be useful to split the training set further to have a validation set:

[44]:
```
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
32768/29515 [==================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [==============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
```

```
datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
```

We will use fashion items classification dataset. It has the same format as which is harder than the digit classification. It has

The training set contains 60,000 grayscale images, each 28x28 pixels:

[45]:
```
X_train_full.shape
```

[45]:
```
(60000, 28, 28)
```

Each pixel intensity is represented as a byte (0 to 255):

[46]:
```
X_train_full.dtype
```

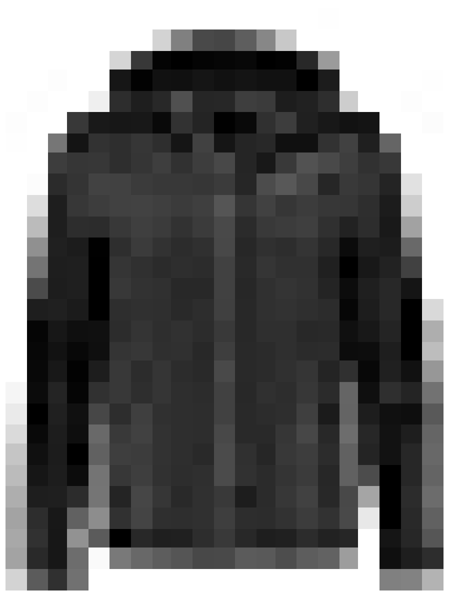
[46]:
```
dtype('uint8')
```

Let's split the full training set into a validation set and a (smaller) training set. We also scale the pixel intensities down to the 0-1 range and convert them to floats, by dividing by 255.

[47]:
```
X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test / 255.
```

You can plot an image using Matplotlib's `imshow()` function, with a `'binary'` color map:

[49]:
```
plt.imshow(X_train[0], cmap="binary")
plt.axis('off')
plt.show()
```



The labels are the class IDs (represented as uint8), from 0 to 9:

[50]:
```
y_train
```

[50]:
```
array([4, 0, 7, …, 3, 0, 5], dtype=uint8)
```

Here are the corresponding class names:

[51]:
```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

So the first image in the training set is a coat:

[52]:
```
class_names[y_train[0]]
```

[52]:
```
'Coat'
```

The validation set contains 5,000 images, and the test set contains 10,000 images:

[53]:
```
X_valid.shape
```

[53]:
```
(5000, 28, 28)
```

[54]:
```
X_test.shape
```

[54]:
```
(10000, 28, 28)
```

Let's take a look at a sample of the images in the dataset:

[55]:
```
n_rows = 4
n_cols = 10
plt.figure(figsize=(n_cols * 1.2, n_rows * 1.2))
for row in range(n_rows):
    for col in range(n_cols):
        index = n_cols * row + col
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(X_train[index], cmap="binary", interpolation="nearest")
        plt.axis('off')
        plt.title(class_names[y_train[index]], fontsize=12)
plt.subplots_adjust(wspace=0.2, hspace=0.5)
save_fig('fashion_mnist_plot', tight_layout=False)
plt.show()
```

```
Saving figure fashion_mnist_plot
```

Let's a build NN model: 784 -> 300 -> 100 -> 10

[56]:
```
# Sequential model
model = keras.models.Sequential()
# input layer 784
model.add(keras.layers.Flatten(input_shape=[28, 28]))
# first hidden layer 300 neurons
model.add(keras.layers.Dense(300, activation="relu"))
# second hidden layer 100 neurons
model.add(keras.layers.Dense(100, activation="relu"))
# output neuron layer:  class with the largest probability
model.add(keras.layers.Dense(10, activation="softmax"))
```

[57]:
```
keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

[58]:
```
# Same model
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

[59]:
```
model.layers
```

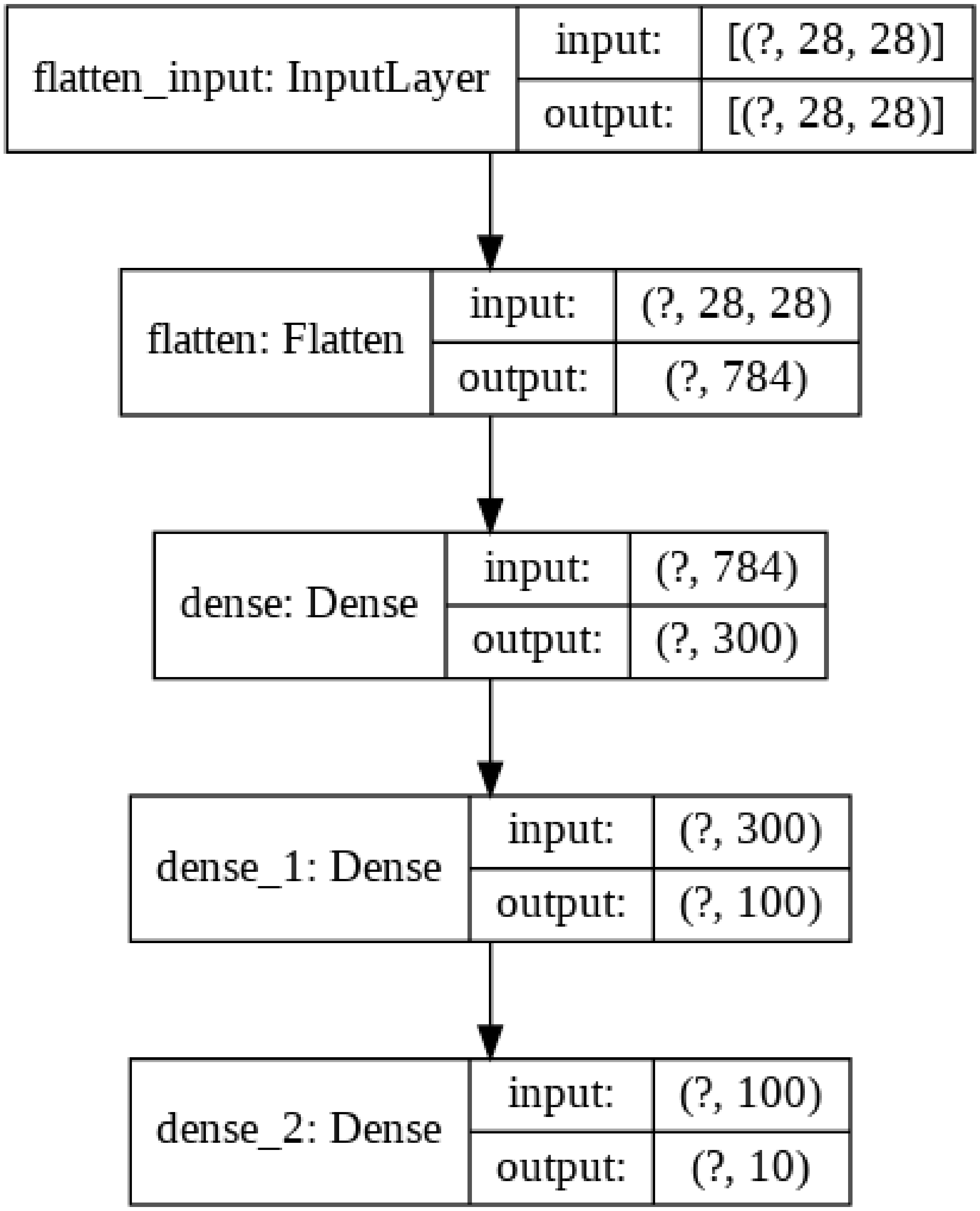
[59]:
```
[<tensorflow.python.keras.layers.core.Flatten at 0x7efc44b0f080>,
 <tensorflow.python.keras.layers.core.Dense at 0x7efc44b0f208>,
 <tensorflow.python.keras.layers.core.Dense at 0x7efc44b0f470>,
 <tensorflow.python.keras.layers.core.Dense at 0x7efc44b0f6d8>]
```

[60]:
```
# Model summary layer shape and the number of parameters
model.summary()
# 784*300 = 235,000 connections
# Very flexible model
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0

_________________________________________________________________
dense (Dense)                (None, 300)               235500

_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100

_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0

_________________________________________________________________
```

[61]:
```
keras.utils.plot_model(model, "my_fashion_mnist_model.png", show_shapes=True)
```

[61]:



[62]:
```
hidden1 = model.layers[1]
hidden1.name
```

[62]:
```
'dense'
```

[64]:
```
model.get_layer(hidden1.name) is hidden1
```

```
[64]: True
```

```
[65]: # Get matrix of initial coefficients and intercepts
      weights, biases = hidden1.get_weights()
```

```
[66]: weights
      # Random initialization of weights
```

```
[66]: array([[ 0.02448617, -0.00877795, -0.02189048, …, -0.02766046,
               0.03859074, -0.06889391],
             [ 0.00476504, -0.03105379, -0.0586676 , …,  0.00602964,
              -0.02763776, -0.04165364],
             [-0.06189284, -0.06901957,  0.07102345, …, -0.04238207,
               0.07121518, -0.07331658],
             …,
             [-0.03048757,  0.02155137, -0.05400612, …, -0.00113463,
               0.00228987,  0.05581069],
             [ 0.07061854, -0.06960931,  0.07038955, …, -0.00384101,
               0.00034875,  0.02878492],
             [-0.06022581,  0.01577859, -0.02585464, …, -0.00527829,
               0.00272203, -0.06793761]], dtype=float32)
```

```
[67]: weights.shape
```

```
[67]: (784, 300)
```

```
[68]: biases
      # Biases are initialized at zero
```

```
[68]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

[69]:
```
biases.shape
```

[69]:
```
(300,)
```

[70]:
```
# estimate the model
# "sparse_categorical_crossentropy" predict one category number
# if instead we wanted a vector of probabilities for eachs class then we would␣
↪choose "categorical_crossenthropy"
# The we will have to do np.argmax() to find the category with the largest␣
↪probability
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
# sgd Keras will perform back-propagation with stochastic gradient descent
# here we don't specify learninng rate , defualt lr=0.01
# For classifier accuracy is a usefull acuracy measure
```

This is equivalent to:

```
model.compile(loss=keras.losses.sparse_categorical_crossentropy,
              optimizer=keras.optimizers.SGD(),
              metrics=[keras.metrics.sparse_categorical_accuracy])
```

[71]:
```
# save history of the model training
# Go through the whole data 30 times
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```

```
Epoch 1/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.7237 -
accuracy: 0.7643 - val_loss: 0.5213 - val_accuracy: 0.8226
Epoch 2/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.4842 -
accuracy: 0.8318 - val_loss: 0.4353 - val_accuracy: 0.8526
Epoch 3/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.4391 -
accuracy: 0.8458 - val_loss: 0.5304 - val_accuracy: 0.7996
Epoch 4/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.4123 -
accuracy: 0.8566 - val_loss: 0.3916 - val_accuracy: 0.8650
Epoch 5/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3939 -
accuracy: 0.8622 - val_loss: 0.3745 - val_accuracy: 0.8690
Epoch 6/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3752 -
accuracy: 0.8675 - val_loss: 0.3718 - val_accuracy: 0.8724
Epoch 7/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3631 -
```

```
accuracy: 0.8716 - val_loss: 0.3616 - val_accuracy: 0.8736
Epoch 8/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3514 -
accuracy: 0.8747 - val_loss: 0.3853 - val_accuracy: 0.8608
Epoch 9/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3412 -
accuracy: 0.8793 - val_loss: 0.3573 - val_accuracy: 0.8718
Epoch 10/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.3317 -
accuracy: 0.8821 - val_loss: 0.3420 - val_accuracy: 0.8786
Epoch 11/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.3238 -
accuracy: 0.8839 - val_loss: 0.3450 - val_accuracy: 0.8770
Epoch 12/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.3147 -
accuracy: 0.8865 - val_loss: 0.3306 - val_accuracy: 0.8832
Epoch 13/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3077 -
accuracy: 0.8896 - val_loss: 0.3274 - val_accuracy: 0.8868
Epoch 14/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.3019 -
accuracy: 0.8917 - val_loss: 0.3420 - val_accuracy: 0.8772
Epoch 15/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2943 -
accuracy: 0.8940 - val_loss: 0.3222 - val_accuracy: 0.8842
Epoch 16/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2889 -
accuracy: 0.8971 - val_loss: 0.3090 - val_accuracy: 0.8906
Epoch 17/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2835 -
accuracy: 0.8979 - val_loss: 0.3546 - val_accuracy: 0.8736
Epoch 18/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2775 -
accuracy: 0.9006 - val_loss: 0.3136 - val_accuracy: 0.8902
Epoch 19/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2726 -
accuracy: 0.9024 - val_loss: 0.3110 - val_accuracy: 0.8904
Epoch 20/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2671 -
accuracy: 0.9036 - val_loss: 0.3271 - val_accuracy: 0.8818
Epoch 21/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2621 -
accuracy: 0.9056 - val_loss: 0.3066 - val_accuracy: 0.8926
Epoch 22/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.2576 -
accuracy: 0.9071 - val_loss: 0.2968 - val_accuracy: 0.8972
Epoch 23/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2533 -
```

```
accuracy: 0.9086 - val_loss: 0.2997 - val_accuracy: 0.8936
Epoch 24/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2484 -
accuracy: 0.9104 - val_loss: 0.3079 - val_accuracy: 0.8890
Epoch 25/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2442 -
accuracy: 0.9127 - val_loss: 0.2977 - val_accuracy: 0.8948
Epoch 26/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2404 -
accuracy: 0.9138 - val_loss: 0.3069 - val_accuracy: 0.8906
Epoch 27/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2360 -
accuracy: 0.9155 - val_loss: 0.3040 - val_accuracy: 0.8940
Epoch 28/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2327 -
accuracy: 0.9166 - val_loss: 0.3003 - val_accuracy: 0.8934
Epoch 29/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2284 -
accuracy: 0.9181 - val_loss: 0.3050 - val_accuracy: 0.8908
Epoch 30/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2248 -
accuracy: 0.9203 - val_loss: 0.3055 - val_accuracy: 0.8934
```

[75]:
```
# final validation accuracy 0.8934, training accuracy 0.9203. They are fairly␣
↪similar -> no overfitting.
history.params
# Each epoch has 1719 steps - when the NN took a small batch of the data and␣
↪run it through the model.
# Defaults batch size is 32 observations. There is not agreement on
```

[75]:
```
{'epochs': 30, 'steps': 1719, 'verbose': 1}
```

If the training data is skewed : some classes are more rare than others, we can set class_weight option in the fit() command.

[75]:
```

```

[76]:
```
print(history.epoch)
```

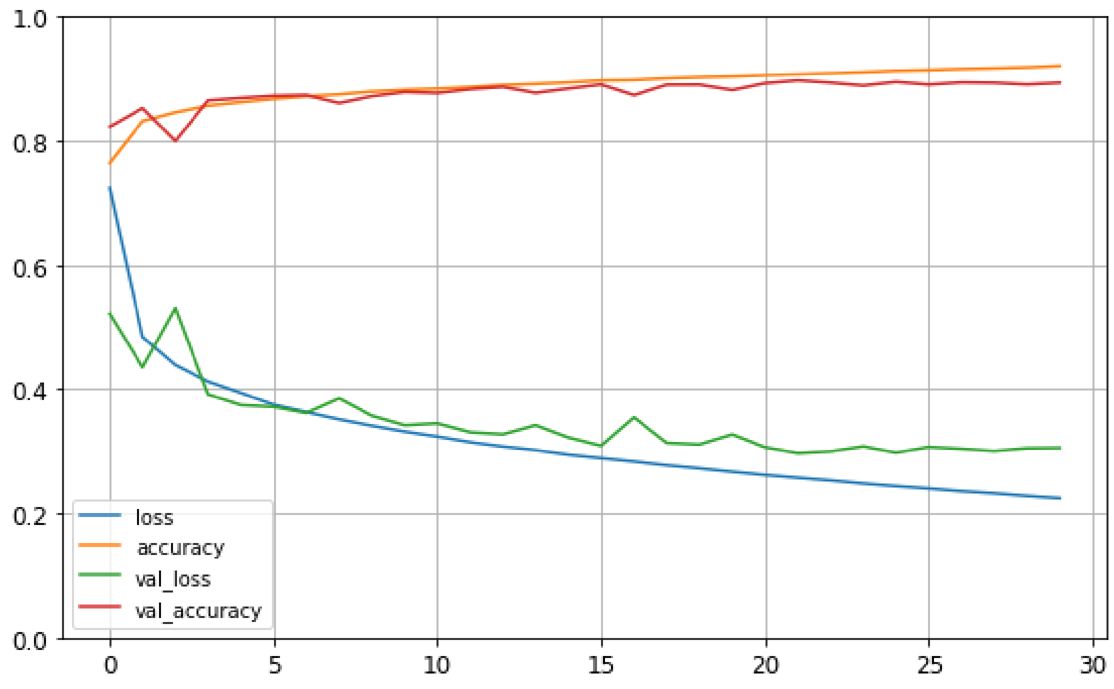
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29]
```

[77]:
```
history.history.keys()
```

[77]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
[78]: pd.DataFrame(history.history).plot(figsize=(8, 5))
      plt.grid(True)
      plt.gca().set_ylim(0, 1)
      save_fig("keras_learning_curves_plot")
      plt.show()
      # History of accuracy.
      # The distnace between trianingn and validation is small, but it starts to grow
      #in the last epochs. The validation accuracy is still increasing, so we should␣
       ↪probably increase traiing.
      # Validation error is computed in the end of the epoch, and training error in␣
       ↪computed during the epoch estimation
```

```
Saving figure keras_learning_curves_plot
```



```
[79]: # Evaluation of model perfomance
      model.evaluate(X_test, y_test)
      # We get lower perfomance on test data. We have not performed any␣
       ↪hyperparameter tuning and this is a bad lack.
```

```
313/313 [==============================] - 0s 2ms/step - loss: 0.3382 -
accuracy: 0.8822
```

```
[79]: [0.3381877839565277, 0.8822000026702881]
```

[80]:
```
# predict class of first 3 observatrions
X_new = X_test[:3]
y_proba = model.predict(X_new)
print(class_names)
y_proba.round(2)
# probability of difference classes.
# First observation is a 96% ankle boot and 3% sneaker
```

```
['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt',
'Sneaker', 'Bag', 'Ankle boot']
```

[80]:
```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.03, 0.  , 0.96],
       [0.  , 0.  , 0.99, 0.  , 0.01, 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]],
      dtype=float32)
```

[81]:
```
# Textbook command is depreciated
y_pred = np.argmax(model.predict(X_new), axis=-1)
y_pred
```

[81]:
```
array([9, 2, 1])
```

[82]:
```
np.array(class_names)[y_pred]
```

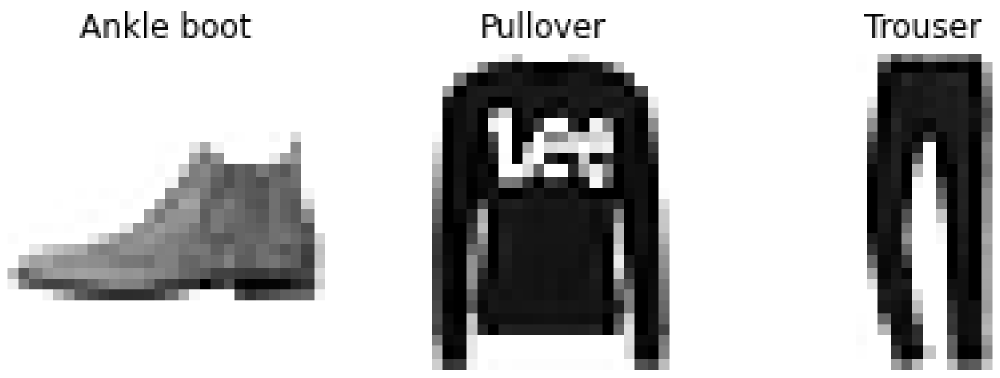
[82]:
```
array(['Ankle boot', 'Pullover', 'Trouser'], dtype='<U11')
```

[83]:
```
y_new = y_test[:3]
y_new
```

[83]:
```
array([9, 2, 1], dtype=uint8)
```

[84]:
```
plt.figure(figsize=(7.2, 2.4))
for index, image in enumerate(X_new):
    plt.subplot(1, 3, index + 1)
    plt.imshow(image, cmap="binary", interpolation="nearest")
    plt.axis('off')
    plt.title(class_names[y_test[index]], fontsize=12)
plt.subplots_adjust(wspace=0.2, hspace=0.5)
save_fig('fashion_mnist_images_plot', tight_layout=False)
plt.show()
```

```
Saving figure fashion_mnist_images_plot
```



8 Regression MLP

Let's load, split and scale the California housing dataset (the original one, not the modified one as in chapter 2). We will use only numeric variables (no *ocean_proximity* feature).

Let's estimate regression. The final outcome is just one number, so the output layer has one neuron.

[85]:
```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

housing = fetch_california_housing()
# split data into training and testing
X_train_full, X_test, y_train_full, y_test = train_test_split(housing.data,␣
↪housing.target, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full,␣
↪y_train_full, random_state=42)
# scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

[85]:
```
```

[86]:
```
np.random.seed(42)
tf.random.set_seed(42)
```

[87]:
```
# start with one input layer 11->30 neurons and one output : 30->1 predicting␣
↪the price of a house
```

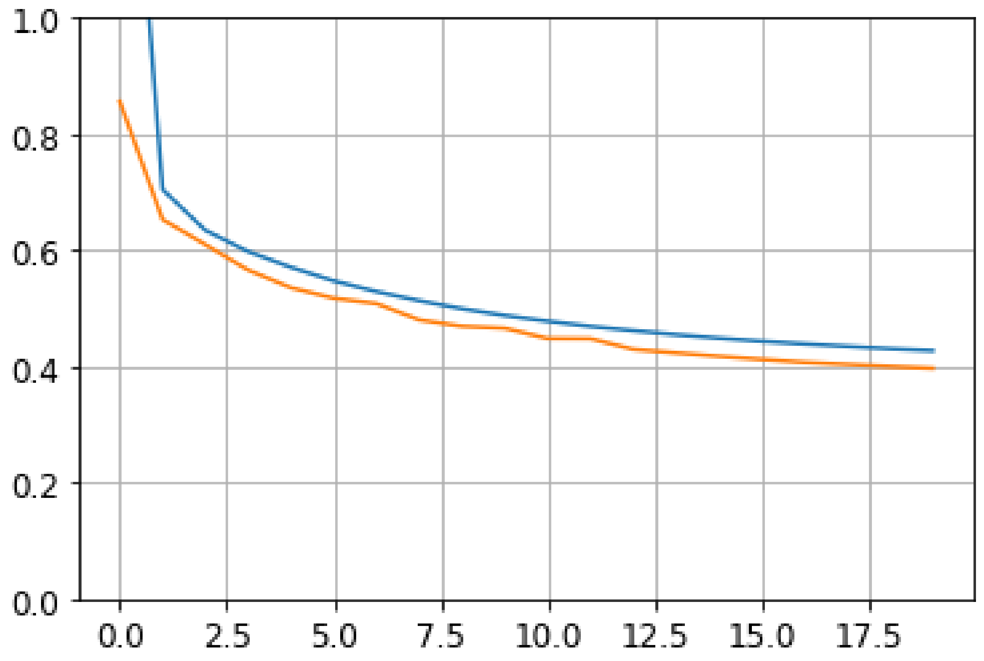
```
# Regression uses loss="mean_squared_error", lr=1e-3 is different from defaults␣
↪0.01
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=X_train.shape[1:]),
    keras.layers.Dense(1)
])
model.compile(loss="mean_squared_error", optimizer=keras.optimizers.
↪SGD(lr=1e-3))
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_valid,␣
↪y_valid))
mse_test = model.evaluate(X_test, y_test)
X_new = X_test[:3]
y_pred = model.predict(X_new)
```

```
Epoch 1/20
363/363 [==============================] - 0s 1ms/step - loss: 1.6419 -
val_loss: 0.8560
Epoch 2/20
363/363 [==============================] - 0s 1ms/step - loss: 0.7047 -
val_loss: 0.6531
Epoch 3/20
363/363 [==============================] - 0s 1ms/step - loss: 0.6345 -
val_loss: 0.6099
Epoch 4/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5977 -
val_loss: 0.5658
Epoch 5/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5706 -
val_loss: 0.5355
Epoch 6/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5472 -
val_loss: 0.5173
Epoch 7/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5288 -
val_loss: 0.5081
Epoch 8/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5130 -
val_loss: 0.4799
Epoch 9/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4992 -
val_loss: 0.4690
Epoch 10/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4875 -
val_loss: 0.4656
Epoch 11/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4777 -
val_loss: 0.4482
```

```
Epoch 12/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4688 -
val_loss: 0.4479
Epoch 13/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4615 -
val_loss: 0.4296
Epoch 14/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4547 -
val_loss: 0.4233
Epoch 15/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4488 -
val_loss: 0.4176
Epoch 16/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4435 -
val_loss: 0.4123
Epoch 17/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4389 -
val_loss: 0.4071
Epoch 18/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4347 -
val_loss: 0.4037
Epoch 19/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4306 -
val_loss: 0.4000
Epoch 20/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4273 -
val_loss: 0.3969
162/162 [==============================] - 0s 760us/step - loss: 0.4212
```

[89]:

```
plt.plot(pd.DataFrame(history.history))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```



9 Functional API

Not all neural network models are simply sequential. Some may have complex topologies. Some may have multiple inputs and/or multiple outputs. For example, a Wide & Deep neural network (see paper) connects all or part of the inputs directly to the output layer.

[90]:
```
np.random.seed(42)
tf.random.set_seed(42)
```

[92]:
```
# set input layer where number of neurons equal to the number of inputs
input_ = keras.layers.Input(shape=X_train.shape[1:])
# 11->30 layer 1
hidden1 = keras.layers.Dense(30, activation="relu")(input_)
# 11->30 layer 2
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
# horizontally concatenated input layer and hidden2 into 11+30 = 41 neurons␣
 ↪layers
concat = keras.layers.concatenate([input_, hidden2])
# one output
output = keras.layers.Dense(1)(concat)
model = keras.models.Model(inputs=[input_], outputs=[output])
```

[93]:
```
model.summary()
```

```
Model: "functional_1"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape          Param #     Connected to
================================================================================
==================
input_1 (InputLayer)            [(None, 8)]           0
________________________________________________________________________________
__________________
dense_5 (Dense)                 (None, 30)            270         input_1[0][0]
________________________________________________________________________________
__________________
dense_6 (Dense)                 (None, 30)            930         dense_5[0][0]
________________________________________________________________________________
__________________
concatenate (Concatenate)       (None, 38)            0           input_1[0][0]
                                                                  dense_6[0][0]
________________________________________________________________________________
__________________
dense_7 (Dense)                 (None, 1)             39
concatenate[0][0]
================================================================================
==================
Total params: 1,239
Trainable params: 1,239
Non-trainable params: 0
________________________________________________________________________________
__________________
```

[94]:
```
# same hypertparameters with different architecture
model.compile(loss="mean_squared_error", optimizer=keras.optimizers.
 ↪SGD(lr=1e-3))
history = model.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))
mse_test = model.evaluate(X_test, y_test)
y_pred = model.predict(X_new)
```

```
Epoch 1/20
363/363 [==============================] - 0s 1ms/step - loss: 1.2611 -
val_loss: 3.3940
Epoch 2/20
363/363 [==============================] - 0s 1ms/step - loss: 0.6580 -
val_loss: 0.9360
Epoch 3/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5878 -
val_loss: 0.5649
Epoch 4/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5582 -
```

```
val_loss: 0.5712
Epoch 5/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5347 -
val_loss: 0.5045
Epoch 6/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5158 -
val_loss: 0.4831
Epoch 7/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5002 -
val_loss: 0.4639
Epoch 8/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4876 -
val_loss: 0.4638
Epoch 9/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4760 -
val_loss: 0.4421
Epoch 10/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4659 -
val_loss: 0.4313
Epoch 11/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4577 -
val_loss: 0.4345
Epoch 12/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4498 -
val_loss: 0.4168
Epoch 13/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4428 -
val_loss: 0.4230
Epoch 14/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4366 -
val_loss: 0.4047
Epoch 15/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4307 -
val_loss: 0.4078
Epoch 16/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4257 -
val_loss: 0.3938
Epoch 17/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4210 -
val_loss: 0.3952
Epoch 18/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4167 -
val_loss: 0.3860
Epoch 19/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4121 -
val_loss: 0.3827
Epoch 20/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4088 -
```

```
val_loss: 0.4054
162/162 [==============================] - 0s 769us/step - loss: 0.4032
```

Results in epoch 19 lower validation loss than in the one-layer simple model (0.3969).

What if you want to send different subsets of input features through the wide or deep paths? We will send 5 features (features 0 to 4), and 6 through the deep path (features 2 to 7). Note that 3 features will go through both (features 2, 3 and 4).

[96]:
```
np.random.seed(42)
tf.random.set_seed(42)
```

[97]:
```
# input that processed through only layer
input_A = keras.layers.Input(shape=[5], name="wide_input")
# input that goes through 3 layers
input_B = keras.layers.Input(shape=[6], name="deep_input")
hidden1 = keras.layers.Dense(30, activation="relu")(input_B)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
# vertically merge deep and wide parts
concat = keras.layers.concatenate([input_A, hidden2])
# output
output = keras.layers.Dense(1, name="output")(concat)
model = keras.models.Model(inputs=[input_A, input_B], outputs=[output])
```

[98]:
```
# compile model
model.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1e-3))
# break X into deep and wide
X_train_A, X_train_B = X_train[:, :5], X_train[:, 2:]
X_valid_A, X_valid_B = X_valid[:, :5], X_valid[:, 2:]
X_test_A, X_test_B = X_test[:, :5], X_test[:, 2:]
X_new_A, X_new_B = X_test_A[:3], X_test_B[:3]
# fit the model
history = model.fit((X_train_A, X_train_B), y_train, epochs=20,
                    validation_data=((X_valid_A, X_valid_B), y_valid))
# evaluate using testing data
mse_test = model.evaluate((X_test_A, X_test_B), y_test)
y_pred = model.predict((X_new_A, X_new_B))
```

```
Epoch 1/20
363/363 [==============================] - 1s 2ms/step - loss: 1.8145 -
val_loss: 0.8072
Epoch 2/20
363/363 [==============================] - 0s 1ms/step - loss: 0.6771 -
val_loss: 0.6658
Epoch 3/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5979 -
val_loss: 0.5687
```

```
Epoch 4/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5584 -
val_loss: 0.5296
Epoch 5/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5334 -
val_loss: 0.4993
Epoch 6/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5120 -
val_loss: 0.4811
Epoch 7/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4970 -
val_loss: 0.4696
Epoch 8/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4843 -
val_loss: 0.4496
Epoch 9/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4730 -
val_loss: 0.4404
Epoch 10/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4644 -
val_loss: 0.4315
Epoch 11/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4570 -
val_loss: 0.4268
Epoch 12/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4510 -
val_loss: 0.4166
Epoch 13/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4462 -
val_loss: 0.4125
Epoch 14/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4421 -
val_loss: 0.4074
Epoch 15/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4385 -
val_loss: 0.4044
Epoch 16/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4356 -
val_loss: 0.4007
Epoch 17/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4322 -
val_loss: 0.4013
Epoch 18/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4305 -
val_loss: 0.3987
Epoch 19/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4274 -
val_loss: 0.3934
```

```
Epoch 20/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4261 -
val_loss: 0.4204
162/162 [==============================] - 0s 1ms/step - loss: 0.4219
WARNING:tensorflow:5 out of the last 6 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7efc46a15b70>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

Adding an auxiliary output for regularization:

Sometimes we need to predict multiple outputs from the same data, for example: image recognition and the classification of the emotional state. Then we use twoo different output vectors.

[100]:
```
np.random.seed(42)
tf.random.set_seed(42)
```

[101]:
```
input_A = keras.layers.Input(shape=[5], name="wide_input")
input_B = keras.layers.Input(shape=[6], name="deep_input")
hidden1 = keras.layers.Dense(30, activation="relu")(input_B)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_A, hidden2])
output = keras.layers.Dense(1, name="main_output")(concat)
# addutional output
aux_output = keras.layers.Dense(1, name="aux_output")(hidden2)
model = keras.models.Model(inputs=[input_A, input_B],
                           # use two outputs in a model
                           outputs=[output, aux_output])
```

[102]:
```
#The model tried to minimize weighted sum of the MSE of the two outpus
model.compile(loss=["mse", "mse"], loss_weights=[0.9, 0.1], optimizer=keras.
 ↪optimizers.SGD(lr=1e-3))
```

[103]:
```
history = model.fit([X_train_A, X_train_B], [y_train, y_train], epochs=20,
                    validation_data=([X_valid_A, X_valid_B], [y_valid,␣
 ↪y_valid]))
```

```
Epoch 1/20
363/363 [==============================] - 1s 2ms/step - loss: 2.1365 -
main_output_loss: 1.9196 - aux_output_loss: 4.0890 - val_loss: 1.6233 -
val_main_output_loss: 0.8468 - val_aux_output_loss: 8.6117
Epoch 2/20
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.8905 -
main_output_loss: 0.6969 - aux_output_loss: 2.6326 - val_loss: 1.5163 -
val_main_output_loss: 0.6836 - val_aux_output_loss: 9.0109
Epoch 3/20
363/363 [==============================] - 0s 1ms/step - loss: 0.7429 -
main_output_loss: 0.6088 - aux_output_loss: 1.9499 - val_loss: 1.4639 -
val_main_output_loss: 0.6229 - val_aux_output_loss: 9.0326
Epoch 4/20
363/363 [==============================] - 1s 1ms/step - loss: 0.6771 -
main_output_loss: 0.5691 - aux_output_loss: 1.6485 - val_loss: 1.3388 -
val_main_output_loss: 0.5481 - val_aux_output_loss: 8.4552
Epoch 5/20
363/363 [==============================] - 1s 1ms/step - loss: 0.6381 -
main_output_loss: 0.5434 - aux_output_loss: 1.4911 - val_loss: 1.2177 -
val_main_output_loss: 0.5194 - val_aux_output_loss: 7.5030
Epoch 6/20
363/363 [==============================] - 1s 1ms/step - loss: 0.6079 -
main_output_loss: 0.5207 - aux_output_loss: 1.3923 - val_loss: 1.0935 -
val_main_output_loss: 0.5106 - val_aux_output_loss: 6.3396
Epoch 7/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5853 -
main_output_loss: 0.5040 - aux_output_loss: 1.3175 - val_loss: 0.9918 -
val_main_output_loss: 0.5115 - val_aux_output_loss: 5.3151
Epoch 8/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5666 -
main_output_loss: 0.4898 - aux_output_loss: 1.2572 - val_loss: 0.8733 -
val_main_output_loss: 0.4733 - val_aux_output_loss: 4.4740
Epoch 9/20
363/363 [==============================] - 1s 1ms/step - loss: 0.5504 -
main_output_loss: 0.4771 - aux_output_loss: 1.2101 - val_loss: 0.7832 -
val_main_output_loss: 0.4555 - val_aux_output_loss: 3.7323
Epoch 10/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5373 -
main_output_loss: 0.4671 - aux_output_loss: 1.1695 - val_loss: 0.7170 -
val_main_output_loss: 0.4604 - val_aux_output_loss: 3.0262
Epoch 11/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5266 -
main_output_loss: 0.4591 - aux_output_loss: 1.1344 - val_loss: 0.6510 -
val_main_output_loss: 0.4293 - val_aux_output_loss: 2.6468
Epoch 12/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5173 -
main_output_loss: 0.4520 - aux_output_loss: 1.1048 - val_loss: 0.6051 -
val_main_output_loss: 0.4310 - val_aux_output_loss: 2.1722
Epoch 13/20
363/363 [==============================] - 0s 1ms/step - loss: 0.5095 -
main_output_loss: 0.4465 - aux_output_loss: 1.0765 - val_loss: 0.5644 -
val_main_output_loss: 0.4161 - val_aux_output_loss: 1.8992
Epoch 14/20
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.5027 -
main_output_loss: 0.4417 - aux_output_loss: 1.0511 - val_loss: 0.5354 -
val_main_output_loss: 0.4119 - val_aux_output_loss: 1.6466
Epoch 15/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4967 -
main_output_loss: 0.4376 - aux_output_loss: 1.0280 - val_loss: 0.5124 -
val_main_output_loss: 0.4047 - val_aux_output_loss: 1.4812
Epoch 16/20
363/363 [==============================] - 1s 1ms/step - loss: 0.4916 -
main_output_loss: 0.4343 - aux_output_loss: 1.0070 - val_loss: 0.4934 -
val_main_output_loss: 0.4034 - val_aux_output_loss: 1.3035
Epoch 17/20
363/363 [==============================] - 1s 1ms/step - loss: 0.4867 -
main_output_loss: 0.4311 - aux_output_loss: 0.9872 - val_loss: 0.4801 -
val_main_output_loss: 0.3984 - val_aux_output_loss: 1.2150
Epoch 18/20
363/363 [==============================] - 1s 1ms/step - loss: 0.4829 -
main_output_loss: 0.4289 - aux_output_loss: 0.9686 - val_loss: 0.4694 -
val_main_output_loss: 0.3962 - val_aux_output_loss: 1.1279
Epoch 19/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4785 -
main_output_loss: 0.4260 - aux_output_loss: 0.9510 - val_loss: 0.4580 -
val_main_output_loss: 0.3936 - val_aux_output_loss: 1.0372
Epoch 20/20
363/363 [==============================] - 0s 1ms/step - loss: 0.4756 -
main_output_loss: 0.4246 - aux_output_loss: 0.9344 - val_loss: 0.4655 -
val_main_output_loss: 0.4048 - val_aux_output_loss: 1.0118
```

[104]:
```
total_loss, main_loss, aux_loss = model.evaluate(
    [X_test_A, X_test_B], [y_test, y_test])
y_pred_main, y_pred_aux = model.predict([X_new_A, X_new_B])
```

```
162/162 [==============================] - 0s 911us/step - loss: 0.4668 -
main_output_loss: 0.4178 - aux_output_loss: 0.9082
WARNING:tensorflow:6 out of the last 7 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7efc45a2ac80>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

[ ]:
```

```

10 Saving and Restoring

[105]:
```
np.random.seed(42)
tf.random.set_seed(42)
```

[106]:
```
# craete a sample model to save
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=[8]),
    keras.layers.Dense(30, activation="relu"),
    keras.layers.Dense(1)
])
```

[107]:
```
model.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1e-3))
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_valid,␣
 ↪y_valid))
mse_test = model.evaluate(X_test, y_test)
```

```
Epoch 1/10
363/363 [==============================] - 1s 1ms/step - loss: 1.8866 -
val_loss: 0.7126
Epoch 2/10
363/363 [==============================] - 0s 1ms/step - loss: 0.6577 -
val_loss: 0.6880
Epoch 3/10
363/363 [==============================] - 0s 1ms/step - loss: 0.5934 -
val_loss: 0.5803
Epoch 4/10
363/363 [==============================] - 0s 1ms/step - loss: 0.5557 -
val_loss: 0.5166
Epoch 5/10
363/363 [==============================] - 0s 1ms/step - loss: 0.5272 -
val_loss: 0.4895
Epoch 6/10
363/363 [==============================] - 0s 1ms/step - loss: 0.5033 -
val_loss: 0.4951
Epoch 7/10
363/363 [==============================] - 0s 1ms/step - loss: 0.4854 -
val_loss: 0.4861
Epoch 8/10
363/363 [==============================] - 0s 1ms/step - loss: 0.4709 -
val_loss: 0.4554
Epoch 9/10
363/363 [==============================] - 0s 1ms/step - loss: 0.4578 -
val_loss: 0.4413
Epoch 10/10
363/363 [==============================] - 0s 1ms/step - loss: 0.4474 -
val_loss: 0.4379
```

```
162/162 [==============================] - 0s 801us/step - loss: 0.4382
```

[108]:
```
# save modle in HDF5 format: architecture and hyperparameters, opmitmizer,␣
↪starting state
model.save("my_keras_model.h5")
```

[109]:
```
# load model
model = keras.models.load_model("my_keras_model.h5")
```

[110]:
```
# predict y from the loaded model
model.predict(X_new)
```

```
WARNING:tensorflow:7 out of the last 8 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7efc45c6c950>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

[110]:
```
array([[0.5400236],
       [1.6505969],
       [3.009824 ]], dtype=float32)
```

[ ]:
```
# We can save and load only weights if needed
model.save_weights("my_keras_weights.ckpt")
```

[ ]:
```
model.load_weights("my_keras_weights.ckpt")
```

[ ]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7ff383d586d8>
```

11 Using Callbacks during Training

[111]:
```
# Model saves multiple checkpoints with different results and we may want to␣
↪restore a particula model state, for example before overfitting started.
keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

[112]:
```
# simple model
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=[8]),
    keras.layers.Dense(30, activation="relu"),
    keras.layers.Dense(1)
])
```

[113]:
```
model.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1e-3))
# Save only best checkpoint. If a better result is found -- override.
checkpoint_cb = keras.callbacks.ModelCheckpoint("my_keras_model.h5",␣
 ↪save_best_only=True)
history = model.fit(X_train, y_train, epochs=10,
                    validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb])
model = keras.models.load_model("my_keras_model.h5") # rollback to best model
mse_test = model.evaluate(X_test, y_test)
```

```
Epoch 1/10
363/363 [==============================] - 1s 1ms/step - loss: 1.8866 -
val_loss: 0.7126
Epoch 2/10
363/363 [==============================] - 0s 1ms/step - loss: 0.6577 -
val_loss: 0.6880
Epoch 3/10
363/363 [==============================] - 0s 1ms/step - loss: 0.5934 -
val_loss: 0.5803
Epoch 4/10
363/363 [==============================] - 0s 1ms/step - loss: 0.5557 -
val_loss: 0.5166
Epoch 5/10
363/363 [==============================] - 0s 1ms/step - loss: 0.5272 -
val_loss: 0.4895
Epoch 6/10
363/363 [==============================] - 0s 1ms/step - loss: 0.5033 -
val_loss: 0.4951
Epoch 7/10
363/363 [==============================] - 0s 1ms/step - loss: 0.4854 -
val_loss: 0.4861
Epoch 8/10
363/363 [==============================] - 0s 1ms/step - loss: 0.4709 -
val_loss: 0.4554
Epoch 9/10
363/363 [==============================] - 0s 1ms/step - loss: 0.4578 -
val_loss: 0.4413
Epoch 10/10
363/363 [==============================] - 0s 1ms/step - loss: 0.4474 -
val_loss: 0.4379
162/162 [==============================] - 0s 763us/step - loss: 0.4382
```

```
[114]: # Another alternative is to stop training if there is not progress. Patience =␣
        ↪10, if 10 iterations with no improvement then stop.
       model.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1e-3))
       early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,
                                                         restore_best_weights=True)
       history = model.fit(X_train, y_train, epochs=100,
                           validation_data=(X_valid, y_valid),
                           callbacks=[checkpoint_cb, early_stopping_cb])
       mse_test = model.evaluate(X_test, y_test)
```

```
Epoch 1/100
363/363 [==============================] - 1s 1ms/step - loss: 0.4393 -
val_loss: 0.4110
Epoch 2/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4315 -
val_loss: 0.4266
Epoch 3/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4259 -
val_loss: 0.3996
Epoch 4/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4201 -
val_loss: 0.3939
Epoch 5/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4154 -
val_loss: 0.3889
Epoch 6/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4111 -
val_loss: 0.3866
Epoch 7/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4074 -
val_loss: 0.3860
Epoch 8/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4040 -
val_loss: 0.3793
Epoch 9/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4008 -
val_loss: 0.3746
Epoch 10/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3976 -
val_loss: 0.3723
Epoch 11/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3950 -
val_loss: 0.3697
Epoch 12/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3923 -
val_loss: 0.3669
Epoch 13/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.3897 -
val_loss: 0.3661
Epoch 14/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3874 -
val_loss: 0.3631
Epoch 15/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3851 -
val_loss: 0.3660
Epoch 16/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3829 -
val_loss: 0.3625
Epoch 17/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3810 -
val_loss: 0.3592
Epoch 18/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3788 -
val_loss: 0.3563
Epoch 19/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3766 -
val_loss: 0.3535
Epoch 20/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3751 -
val_loss: 0.3709
Epoch 21/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3732 -
val_loss: 0.3512
Epoch 22/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3715 -
val_loss: 0.3699
Epoch 23/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3700 -
val_loss: 0.3476
Epoch 24/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3685 -
val_loss: 0.3561
Epoch 25/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3671 -
val_loss: 0.3527
Epoch 26/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3658 -
val_loss: 0.3701
Epoch 27/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3647 -
val_loss: 0.3432
Epoch 28/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3635 -
val_loss: 0.3592
Epoch 29/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.3625 -
val_loss: 0.3521
Epoch 30/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3613 -
val_loss: 0.3626
Epoch 31/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3601 -
val_loss: 0.3431
Epoch 32/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3589 -
val_loss: 0.3766
Epoch 33/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3584 -
val_loss: 0.3374
Epoch 34/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3572 -
val_loss: 0.3407
Epoch 35/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3563 -
val_loss: 0.3614
Epoch 36/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3555 -
val_loss: 0.3348
Epoch 37/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3546 -
val_loss: 0.3573
Epoch 38/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3538 -
val_loss: 0.3367
Epoch 39/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3530 -
val_loss: 0.3425
Epoch 40/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3523 -
val_loss: 0.3369
Epoch 41/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3515 -
val_loss: 0.3514
Epoch 42/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3511 -
val_loss: 0.3427
Epoch 43/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3500 -
val_loss: 0.3678
Epoch 44/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3496 -
val_loss: 0.3563
Epoch 45/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.3490 -
val_loss: 0.3336
Epoch 46/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3481 -
val_loss: 0.3457
Epoch 47/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3478 -
val_loss: 0.3433
Epoch 48/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3471 -
val_loss: 0.3658
Epoch 49/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3466 -
val_loss: 0.3286
Epoch 50/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3460 -
val_loss: 0.3268
Epoch 51/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3454 -
val_loss: 0.3438
Epoch 52/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3449 -
val_loss: 0.3263
Epoch 53/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3444 -
val_loss: 0.3909
Epoch 54/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3439 -
val_loss: 0.3275
Epoch 55/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3435 -
val_loss: 0.3560
Epoch 56/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3430 -
val_loss: 0.3237
Epoch 57/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3423 -
val_loss: 0.3242
Epoch 58/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3419 -
val_loss: 0.3764
Epoch 59/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3417 -
val_loss: 0.3289
Epoch 60/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3410 -
val_loss: 0.3502
Epoch 61/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.3404 -
val_loss: 0.3457
Epoch 62/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3402 -
val_loss: 0.3444
Epoch 63/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3392 -
val_loss: 0.3290
Epoch 64/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3393 -
val_loss: 0.3217
Epoch 65/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3387 -
val_loss: 0.3351
Epoch 66/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3383 -
val_loss: 0.3232
Epoch 67/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3376 -
val_loss: 0.3568
Epoch 68/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3375 -
val_loss: 0.3257
Epoch 69/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3370 -
val_loss: 0.3349
Epoch 70/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3365 -
val_loss: 0.3560
Epoch 71/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3361 -
val_loss: 0.3581
Epoch 72/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3357 -
val_loss: 0.3287
Epoch 73/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3351 -
val_loss: 0.3202
Epoch 74/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3350 -
val_loss: 0.3840
Epoch 75/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3347 -
val_loss: 0.3233
Epoch 76/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3342 -
val_loss: 0.3475
Epoch 77/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.3338 -
val_loss: 0.3408
Epoch 78/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3335 -
val_loss: 0.3461
Epoch 79/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3332 -
val_loss: 0.3348
Epoch 80/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3329 -
val_loss: 0.3355
Epoch 81/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3324 -
val_loss: 0.3276
Epoch 82/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3320 -
val_loss: 0.3167
Epoch 83/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3317 -
val_loss: 0.3279
Epoch 84/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3312 -
val_loss: 0.3637
Epoch 85/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3310 -
val_loss: 0.3174
Epoch 86/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3308 -
val_loss: 0.3155
Epoch 87/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3305 -
val_loss: 0.3529
Epoch 88/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3299 -
val_loss: 0.3256
Epoch 89/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3294 -
val_loss: 0.3630
Epoch 90/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3296 -
val_loss: 0.3383
Epoch 91/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3291 -
val_loss: 0.3212
Epoch 92/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3287 -
val_loss: 0.3458
Epoch 93/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.3285 -
val_loss: 0.3159
Epoch 94/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3281 -
val_loss: 0.3410
Epoch 95/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3276 -
val_loss: 0.3381
Epoch 96/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3273 -
val_loss: 0.3214
162/162 [==============================] - 0s 740us/step - loss: 0.3310
```

[115]:
```
# Epochs 86 has the best results with val_loss = 0.3155, so the training␣
↪continued until epoch 96, when the patience run out
```

12 TensorBoard

[119]:
```
# Tensor board is a tool to visualization of model results
root_logdir = os.path.join(os.curdir, "my_logs")
```

[120]:
```
# get logs
def get_run_logdir():
    import time
    run_id = time.strftime("run_%Y_%m_%d-%H_%M_%S")
    return os.path.join(root_logdir, run_id)

run_logdir = get_run_logdir()
run_logdir
```

[120]:
```
'./my_logs/run_2020_08_15-15_21_51'
```

[122]:
```
# clear session
keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

[123]:
```
# run simple model
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=[8]),
    keras.layers.Dense(30, activation="relu"),
    keras.layers.Dense(1)
])
model.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1e-3))
```

[126]:
```
tensorboard_cb = keras.callbacks.TensorBoard(run_logdir)
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb, tensorboard_cb])
```

```
Epoch 1/30
  1/363 […] - ETA: 0s - loss:
7.8215WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/summary_ops_v2.py:1277: stop (from
tensorflow.python.eager.profiler) is deprecated and will be removed after
2020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the
batch time (batch time: 0.0058s vs `on_train_batch_end` time: 0.0353s). Check
your callbacks.
363/363 [==============================] - 1s 2ms/step - loss: 1.8866 -
val_loss: 0.7126
Epoch 2/30
363/363 [==============================] - 0s 1ms/step - loss: 0.6577 -
val_loss: 0.6880
Epoch 3/30
363/363 [==============================] - 0s 1ms/step - loss: 0.5934 -
val_loss: 0.5803
Epoch 4/30
363/363 [==============================] - 0s 1ms/step - loss: 0.5557 -
val_loss: 0.5166
Epoch 5/30
363/363 [==============================] - 0s 1ms/step - loss: 0.5272 -
val_loss: 0.4895
Epoch 6/30
363/363 [==============================] - 0s 1ms/step - loss: 0.5033 -
val_loss: 0.4951
Epoch 7/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4854 -
val_loss: 0.4861
Epoch 8/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4709 -
val_loss: 0.4554
Epoch 9/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4578 -
val_loss: 0.4413
Epoch 10/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4474 -
val_loss: 0.4379
Epoch 11/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4393 -
```

```
val_loss: 0.4396
Epoch 12/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4318 -
val_loss: 0.4507
Epoch 13/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4261 -
val_loss: 0.3997
Epoch 14/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4202 -
val_loss: 0.3956
Epoch 15/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4155 -
val_loss: 0.3916
Epoch 16/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4112 -
val_loss: 0.3937
Epoch 17/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4077 -
val_loss: 0.3809
Epoch 18/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4040 -
val_loss: 0.3793
Epoch 19/30
363/363 [==============================] - 0s 1ms/step - loss: 0.4004 -
val_loss: 0.3850
Epoch 20/30
363/363 [==============================] - 0s 1ms/step - loss: 0.3980 -
val_loss: 0.3809
Epoch 21/30
363/363 [==============================] - 0s 1ms/step - loss: 0.3949 -
val_loss: 0.3701
Epoch 22/30
363/363 [==============================] - 0s 1ms/step - loss: 0.3924 -
val_loss: 0.3781
Epoch 23/30
363/363 [==============================] - 0s 1ms/step - loss: 0.3898 -
val_loss: 0.3650
Epoch 24/30
363/363 [==============================] - 0s 1ms/step - loss: 0.3874 -
val_loss: 0.3655
Epoch 25/30
363/363 [==============================] - 0s 1ms/step - loss: 0.3851 -
val_loss: 0.3611
Epoch 26/30
363/363 [==============================] - 0s 1ms/step - loss: 0.3829 -
val_loss: 0.3626
Epoch 27/30
363/363 [==============================] - 0s 1ms/step - loss: 0.3809 -
```

```
val_loss: 0.3564
Epoch 28/30
363/363 [==============================] - 0s 1ms/step - loss: 0.3788 -
val_loss: 0.3579
Epoch 29/30
363/363 [==============================] - 0s 1ms/step - loss: 0.3769 -
val_loss: 0.3561
Epoch 30/30
363/363 [==============================] - 0s 1ms/step - loss: 0.3750 -
val_loss: 0.3548
```

To start the TensorBoard server, one option is to open a terminal, if needed activate the virtualenv where you installed TensorBoard, go to this notebook's directory, then type:

```
$ tensorboard --logdir=./my_logs --port=6006
```

You can then open your web browser to localhost:6006 and use TensorBoard. Once you are done, press Ctrl-C in the terminal window, this will shutdown the TensorBoard server.

Alternatively, you can load TensorBoard's Jupyter extension and run it like this:

[128]:
```
%load_ext tensorboard
%tensorboard --logdir=./my_logs --port=6006
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard

Reusing TensorBoard on port 6006 (pid 2255), started 0:00:07 ago. (Use '!kill 2255' to kill it


<IPython.core.display.Javascript object>
```

So far we learned -- How to use Keras in estimation of classification and regression. -- How to build complex neural network with multiple-outputs.

13 Hyperparameter Tuning

[134]:
```
keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

One of the hardest and most important tasks in the estimation of NN is the tuning of hyperparameters. Usually there are many possible combinations, and we have time/resources to try few of them.

14 Fine-Tuning Neural Networks:

1. There are lot hyperparemeters to estimate. Use randomized grid search to narrow-down the parameters space.
2. Complex relationships are better approximated by a lot of layers. If the network has many layers it may not need many nodes.
3. Try to add new layers if it imroves the fit.
4. Recycle lower layers from the similar models
5. Build your network as a funnel between features and outputs. Lower levels tend to have more neurons than upper one.
6. ReLU is good default activation function for classification. Regression problems don't need an activation function.

[ ]:

```

```

[135]:

```
# Model build function to simple model. Inputs number of hidden lyaers, each␣
↪layer has the same number of neurons
def build_model(n_hidden=1, n_neurons=30, learning_rate=3e-3, input_shape=[8]):
    model = keras.models.Sequential()
    # input layer
    model.add(keras.layers.InputLayer(input_shape=input_shape))
    # sequentially add layhers
    for layer in range(n_hidden):
        model.add(keras.layers.Dense(n_neurons, activation="relu"))
        # output layer 1
    model.add(keras.layers.Dense(1))
    optimizer = keras.optimizers.SGD(lr=learning_rate)
    model.compile(loss="mse", optimizer=optimizer)
    return model
```

[136]:

```
# Create a regression function
keras_reg = keras.wrappers.scikit_learn.KerasRegressor(build_model)
```

[137]:

```
# Use early stopping
keras_reg.fit(X_train, y_train, epochs=100,
              validation_data=(X_valid, y_valid),
              callbacks=[keras.callbacks.EarlyStopping(patience=10)])
```

```
Epoch 1/100
363/363 [==============================] - 0s 1ms/step - loss: 1.0896 -
val_loss: 20.7721
Epoch 2/100
363/363 [==============================] - 0s 1ms/step - loss: 0.7606 -
val_loss: 5.0266
Epoch 3/100
363/363 [==============================] - 0s 1ms/step - loss: 0.5456 -
val_loss: 0.5490
Epoch 4/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.4732 -
val_loss: 0.4529
Epoch 5/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4503 -
val_loss: 0.4188
Epoch 6/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4338 -
val_loss: 0.4129
Epoch 7/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4241 -
val_loss: 0.4004
Epoch 8/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4168 -
val_loss: 0.3944
Epoch 9/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4108 -
val_loss: 0.3961
Epoch 10/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4060 -
val_loss: 0.4071
Epoch 11/100
363/363 [==============================] - 0s 1ms/step - loss: 0.4021 -
val_loss: 0.3855
Epoch 12/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3984 -
val_loss: 0.4136
Epoch 13/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3951 -
val_loss: 0.3997
Epoch 14/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3921 -
val_loss: 0.3818
Epoch 15/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3894 -
val_loss: 0.3829
Epoch 16/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3869 -
val_loss: 0.3739
Epoch 17/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3848 -
val_loss: 0.4022
Epoch 18/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3829 -
val_loss: 0.3873
Epoch 19/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3807 -
val_loss: 0.3768
Epoch 20/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.3791 -
val_loss: 0.4191
Epoch 21/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3774 -
val_loss: 0.3927
Epoch 22/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3756 -
val_loss: 0.4237
Epoch 23/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3742 -
val_loss: 0.3523
Epoch 24/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3725 -
val_loss: 0.3842
Epoch 25/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3710 -
val_loss: 0.4162
Epoch 26/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3700 -
val_loss: 0.3980
Epoch 27/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3691 -
val_loss: 0.3474
Epoch 28/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3677 -
val_loss: 0.3920
Epoch 29/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3670 -
val_loss: 0.3566
Epoch 30/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3653 -
val_loss: 0.4191
Epoch 31/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3647 -
val_loss: 0.3721
Epoch 32/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3633 -
val_loss: 0.3948
Epoch 33/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3632 -
val_loss: 0.3423
Epoch 34/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3617 -
val_loss: 0.3453
Epoch 35/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3610 -
val_loss: 0.4068
Epoch 36/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.3608 -
val_loss: 0.3417
Epoch 37/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3596 -
val_loss: 0.3787
Epoch 38/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3589 -
val_loss: 0.3379
Epoch 39/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3582 -
val_loss: 0.3419
Epoch 40/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3572 -
val_loss: 0.3705
Epoch 41/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3570 -
val_loss: 0.3659
Epoch 42/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3563 -
val_loss: 0.3803
Epoch 43/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3552 -
val_loss: 0.3765
Epoch 44/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3548 -
val_loss: 0.3813
Epoch 45/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3543 -
val_loss: 0.3326
Epoch 46/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3532 -
val_loss: 0.3385
Epoch 47/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3527 -
val_loss: 0.3655
Epoch 48/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3521 -
val_loss: 0.3579
Epoch 49/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3525 -
val_loss: 0.3360
Epoch 50/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3510 -
val_loss: 0.3317
Epoch 51/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3504 -
val_loss: 0.3562
Epoch 52/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.3502 -
val_loss: 0.3521
Epoch 53/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3496 -
val_loss: 0.4579
Epoch 54/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3497 -
val_loss: 0.3809
Epoch 55/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3490 -
val_loss: 0.3540
Epoch 56/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3485 -
val_loss: 0.3725
Epoch 57/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3478 -
val_loss: 0.3337
Epoch 58/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3469 -
val_loss: 0.4011
Epoch 59/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3476 -
val_loss: 0.3263
Epoch 60/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3466 -
val_loss: 0.3271
Epoch 61/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3453 -
val_loss: 0.3349
Epoch 62/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3454 -
val_loss: 0.3541
Epoch 63/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3445 -
val_loss: 0.3428
Epoch 64/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3451 -
val_loss: 0.3280
Epoch 65/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3437 -
val_loss: 0.3292
Epoch 66/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3431 -
val_loss: 0.3301
Epoch 67/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3428 -
val_loss: 0.3254
Epoch 68/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.3423 -
val_loss: 0.3245
Epoch 69/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3419 -
val_loss: 0.3255
Epoch 70/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3413 -
val_loss: 0.3666
Epoch 71/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3414 -
val_loss: 0.3370
Epoch 72/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3405 -
val_loss: 0.3267
Epoch 73/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3400 -
val_loss: 0.3245
Epoch 74/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3402 -
val_loss: 0.3663
Epoch 75/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3397 -
val_loss: 0.3290
Epoch 76/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3395 -
val_loss: 0.3235
Epoch 77/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3383 -
val_loss: 0.3386
Epoch 78/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3384 -
val_loss: 0.3362
Epoch 79/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3384 -
val_loss: 0.3222
Epoch 80/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3376 -
val_loss: 0.3644
Epoch 81/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3384 -
val_loss: 0.3420
Epoch 82/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3371 -
val_loss: 0.3253
Epoch 83/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3368 -
val_loss: 0.3246
Epoch 84/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.3362 -
val_loss: 0.3953
Epoch 85/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3372 -
val_loss: 0.3415
Epoch 86/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3359 -
val_loss: 0.3190
Epoch 87/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3356 -
val_loss: 0.3277
Epoch 88/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3351 -
val_loss: 0.3295
Epoch 89/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3348 -
val_loss: 0.3247
Epoch 90/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3344 -
val_loss: 0.3281
Epoch 91/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3341 -
val_loss: 0.3201
Epoch 92/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3338 -
val_loss: 0.3393
Epoch 93/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3336 -
val_loss: 0.3170
Epoch 94/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3334 -
val_loss: 0.3526
Epoch 95/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3329 -
val_loss: 0.4813
Epoch 96/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3339 -
val_loss: 0.3465
Epoch 97/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3324 -
val_loss: 0.4632
Epoch 98/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3333 -
val_loss: 0.6725
Epoch 99/100
363/363 [==============================] - 0s 1ms/step - loss: 0.3330 -
val_loss: 0.5924
Epoch 100/100
```

```
363/363 [==============================] - 0s 1ms/step - loss: 0.3343 -
val_loss: 0.5271
```

```
[137]: <tensorflow.python.keras.callbacks.History at 0x7efc46b74ac8>
```

```
[138]: mse_test = keras_reg.score(X_test, y_test)
       # Accuracy without tuning
```

```
162/162 [==============================] - 0s 841us/step - loss: 0.3346
```

```
[139]: y_pred = keras_reg.predict(X_new)
```

```
WARNING:tensorflow:8 out of the last 9 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7efc46a15158>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to https://ww
w.tensorflow.org/tutorials/customization/performance#python_or_tensor_args and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
[140]: np.random.seed(42)
       tf.random.set_seed(42)
```

Warning: the following cell crashes at the end of training. This seems to be caused by Keras issue #13586, which was triggered by a recent change in Scikit-Learn. Pull Request #13598 seems to fix the issue, so this problem should be resolved soon.

```
[ ]: from scipy.stats import reciprocal
     # Randomized search of hyperparameterrs
     from sklearn.model_selection import RandomizedSearchCV

     param_distribs = {
       # number of hidden layers
         "n_hidden": [0, 1, 2, 3],
         # number of neurons in each layer
         "n_neurons": np.arange(1, 100),
         # learning rate
         "learning_rate": reciprocal(3e-4, 3e-2),
     }
     # 10 random combinations of these parameters
     rnd_search_cv = RandomizedSearchCV(keras_reg, param_distribs, n_iter=10, cv=3,␣
      ↪verbose=2)
     rnd_search_cv.fit(X_train, y_train, epochs=100,
                       validation_data=(X_valid, y_valid),
                    callbacks=[keras.callbacks.EarlyStopping(patience=10)])
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[CV] learning_rate=0.00037192261022352417, n_hidden=3, n_neurons=80 ..
Epoch 1/100

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

242/242 [==============================] - 0s 2ms/step - loss: 3.3520 -
val_loss: 4.4563
Epoch 2/100
242/242 [==============================] - 0s 1ms/step - loss: 1.4974 -
val_loss: 3.1074
Epoch 3/100
242/242 [==============================] - 0s 1ms/step - loss: 1.1100 -
val_loss: 1.6834
Epoch 4/100
242/242 [==============================] - 0s 1ms/step - loss: 0.9274 -
val_loss: 1.0210
Epoch 5/100
242/242 [==============================] - 0s 1ms/step - loss: 0.8229 -
val_loss: 0.7946
Epoch 6/100
242/242 [==============================] - 0s 1ms/step - loss: 0.7590 -
val_loss: 0.7180
Epoch 7/100
242/242 [==============================] - 0s 1ms/step - loss: 0.7177 -
val_loss: 0.6944
Epoch 8/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6893 -
val_loss: 0.6736
Epoch 9/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6675 -
val_loss: 0.6516
Epoch 10/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6490 -
val_loss: 0.6364
Epoch 11/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6329 -
val_loss: 0.6186
Epoch 12/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6183 -
val_loss: 0.6007
Epoch 13/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6045 -
val_loss: 0.5884
Epoch 14/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5917 -
```

```
val_loss: 0.5689
Epoch 15/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5797 -
val_loss: 0.5567
Epoch 16/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5681 -
val_loss: 0.5437
Epoch 17/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5573 -
val_loss: 0.5305
Epoch 18/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5469 -
val_loss: 0.5208
Epoch 19/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5370 -
val_loss: 0.5094
Epoch 20/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5278 -
val_loss: 0.5021
Epoch 21/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5190 -
val_loss: 0.4901
Epoch 22/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5105 -
val_loss: 0.4823
Epoch 23/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5026 -
val_loss: 0.4738
Epoch 24/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4949 -
val_loss: 0.4680
Epoch 25/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4880 -
val_loss: 0.4590
Epoch 26/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4811 -
val_loss: 0.4520
Epoch 27/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4748 -
val_loss: 0.4458
Epoch 28/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4687 -
val_loss: 0.4398
Epoch 29/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4629 -
val_loss: 0.4346
Epoch 30/100
242/242 [==============================] - 0s 2ms/step - loss: 0.4576 -
```

```
val_loss: 0.4293
Epoch 31/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4524 -
val_loss: 0.4241
Epoch 32/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4477 -
val_loss: 0.4206
Epoch 33/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4432 -
val_loss: 0.4167
Epoch 34/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4388 -
val_loss: 0.4130
Epoch 35/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4349 -
val_loss: 0.4101
Epoch 36/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4312 -
val_loss: 0.4065
Epoch 37/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4277 -
val_loss: 0.4033
Epoch 38/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4243 -
val_loss: 0.4020
Epoch 39/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4212 -
val_loss: 0.3975
Epoch 40/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4183 -
val_loss: 0.3986
Epoch 41/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4154 -
val_loss: 0.3972
Epoch 42/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4127 -
val_loss: 0.3940
Epoch 43/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4103 -
val_loss: 0.3895
Epoch 44/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4078 -
val_loss: 0.3940
Epoch 45/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4057 -
val_loss: 0.3901
Epoch 46/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4034 -
```

```
val_loss: 0.3856
Epoch 47/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4014 -
val_loss: 0.3883
Epoch 48/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3994 -
val_loss: 0.3843
Epoch 49/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3975 -
val_loss: 0.3833
Epoch 50/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3956 -
val_loss: 0.3844
Epoch 51/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3937 -
val_loss: 0.3799
Epoch 52/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3922 -
val_loss: 0.3827
Epoch 53/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3905 -
val_loss: 0.3824
Epoch 54/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3889 -
val_loss: 0.3759
Epoch 55/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3871 -
val_loss: 0.3798
Epoch 56/100
242/242 [==============================] - 0s 2ms/step - loss: 0.3859 -
val_loss: 0.3794
Epoch 57/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3844 -
val_loss: 0.3805
Epoch 58/100
242/242 [==============================] - 0s 2ms/step - loss: 0.3829 -
val_loss: 0.3819
Epoch 59/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3816 -
val_loss: 0.3752
Epoch 60/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3802 -
val_loss: 0.3732
Epoch 61/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3792 -
val_loss: 0.3749
Epoch 62/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3778 -
```

```
val_loss: 0.3810
Epoch 63/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3766 -
val_loss: 0.3722
Epoch 64/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3755 -
val_loss: 0.3787
Epoch 65/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3743 -
val_loss: 0.3806
Epoch 66/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3733 -
val_loss: 0.3749
Epoch 67/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3721 -
val_loss: 0.3725
Epoch 68/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3711 -
val_loss: 0.3758
Epoch 69/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3700 -
val_loss: 0.3725
Epoch 70/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3690 -
val_loss: 0.3688
Epoch 71/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3681 -
val_loss: 0.3774
Epoch 72/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3670 -
val_loss: 0.3727
Epoch 73/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3661 -
val_loss: 0.3662
Epoch 74/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3652 -
val_loss: 0.3695
Epoch 75/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3643 -
val_loss: 0.3679
Epoch 76/100
242/242 [==============================] - 0s 2ms/step - loss: 0.3633 -
val_loss: 0.3683
Epoch 77/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3627 -
val_loss: 0.3621
Epoch 78/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3617 -
```

```
val_loss: 0.3698
Epoch 79/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3609 -
val_loss: 0.3677
Epoch 80/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3600 -
val_loss: 0.3613
Epoch 81/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3591 -
val_loss: 0.3684
Epoch 82/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3585 -
val_loss: 0.3684
Epoch 83/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3577 -
val_loss: 0.3619
Epoch 84/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3569 -
val_loss: 0.3677
Epoch 85/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3560 -
val_loss: 0.3598
Epoch 86/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3551 -
val_loss: 0.3704
Epoch 87/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3548 -
val_loss: 0.3587
Epoch 88/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3540 -
val_loss: 0.3581
Epoch 89/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3533 -
val_loss: 0.3596
Epoch 90/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3525 -
val_loss: 0.3697
Epoch 91/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3521 -
val_loss: 0.3587
Epoch 92/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3514 -
val_loss: 0.3563
Epoch 93/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3507 -
val_loss: 0.3600
Epoch 94/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3501 -
```

```
val_loss: 0.3555
Epoch 95/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3495 -
val_loss: 0.3572
Epoch 96/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3486 -
val_loss: 0.3568
Epoch 97/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3482 -
val_loss: 0.3469
Epoch 98/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3476 -
val_loss: 0.3482
Epoch 99/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3467 -
val_loss: 0.3700
Epoch 100/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3465 -
val_loss: 0.3523
121/121 [==============================] - 0s 822us/step - loss: 0.3644
[CV]  learning_rate=0.00037192261022352417, n_hidden=3, n_neurons=80, total=
34.8s
[CV] learning_rate=0.00037192261022352417, n_hidden=3, n_neurons=80 ..
Epoch 1/100

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   34.8s remaining:    0.0s

242/242 [==============================] - 0s 2ms/step - loss: 3.3823 -
val_loss: 5.4433
Epoch 2/100
242/242 [==============================] - 0s 1ms/step - loss: 1.4508 -
val_loss: 7.1226
Epoch 3/100
242/242 [==============================] - 0s 1ms/step - loss: 1.0963 -
val_loss: 6.2038
Epoch 4/100
242/242 [==============================] - 0s 2ms/step - loss: 0.9252 -
val_loss: 5.1899
Epoch 5/100
242/242 [==============================] - 0s 1ms/step - loss: 0.8177 -
val_loss: 4.2648
Epoch 6/100
242/242 [==============================] - 0s 1ms/step - loss: 0.7491 -
val_loss: 3.5610
Epoch 7/100
242/242 [==============================] - 0s 1ms/step - loss: 0.7041 -
val_loss: 2.9506
Epoch 8/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6724 -
```

```
val_loss: 2.4059
Epoch 9/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6483 -
val_loss: 2.0150
Epoch 10/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6283 -
val_loss: 1.7037
Epoch 11/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6109 -
val_loss: 1.4691
Epoch 12/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5952 -
val_loss: 1.2354
Epoch 13/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5810 -
val_loss: 1.0597
Epoch 14/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5679 -
val_loss: 0.9400
Epoch 15/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5556 -
val_loss: 0.8086
Epoch 16/100
242/242 [==============================] - 0s 2ms/step - loss: 0.5441 -
val_loss: 0.7234
Epoch 17/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5333 -
val_loss: 0.6409
Epoch 18/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5232 -
val_loss: 0.5848
Epoch 19/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5137 -
val_loss: 0.5405
Epoch 20/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5049 -
val_loss: 0.5118
Epoch 21/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4968 -
val_loss: 0.4877
Epoch 22/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4891 -
val_loss: 0.4752
Epoch 23/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4821 -
val_loss: 0.4659
Epoch 24/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4756 -
```

```
val_loss: 0.4604
Epoch 25/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4695 -
val_loss: 0.4588
Epoch 26/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4639 -
val_loss: 0.4598
Epoch 27/100
242/242 [==============================] - 0s 2ms/step - loss: 0.4586 -
val_loss: 0.4622
Epoch 28/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4538 -
val_loss: 0.4664
Epoch 29/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4489 -
val_loss: 0.4661
Epoch 30/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4452 -
val_loss: 0.4740
Epoch 31/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4410 -
val_loss: 0.4832
Epoch 32/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4374 -
val_loss: 0.4860
Epoch 33/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4340 -
val_loss: 0.4891
Epoch 34/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4307 -
val_loss: 0.4868
Epoch 35/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4276 -
val_loss: 0.4866
121/121 [==============================] - 0s 788us/step - loss: 0.4431
[CV]  learning_rate=0.00037192261022352417, n_hidden=3, n_neurons=80, total=
12.8s
[CV] learning_rate=0.00037192261022352417, n_hidden=3, n_neurons=80 ..
Epoch 1/100
242/242 [==============================] - 0s 2ms/step - loss: 3.1501 -
val_loss: 3.1013
Epoch 2/100
242/242 [==============================] - 0s 1ms/step - loss: 1.3040 -
val_loss: 2.1131
Epoch 3/100
242/242 [==============================] - 0s 2ms/step - loss: 0.9474 -
val_loss: 1.0709
Epoch 4/100
```

```
242/242 [==============================] - 0s 1ms/step - loss: 0.8352 -
val_loss: 0.8160
Epoch 5/100
242/242 [==============================] - 0s 1ms/step - loss: 0.7814 -
val_loss: 0.7517
Epoch 6/100
242/242 [==============================] - 0s 1ms/step - loss: 0.7477 -
val_loss: 0.7235
Epoch 7/100
242/242 [==============================] - 0s 1ms/step - loss: 0.7214 -
val_loss: 0.7009
Epoch 8/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6994 -
val_loss: 0.7169
Epoch 9/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6804 -
val_loss: 0.6731
Epoch 10/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6621 -
val_loss: 0.6412
Epoch 11/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6449 -
val_loss: 0.6370
Epoch 12/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6294 -
val_loss: 0.6168
Epoch 13/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6146 -
val_loss: 0.5932
Epoch 14/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6005 -
val_loss: 0.5811
Epoch 15/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5871 -
val_loss: 0.5670
Epoch 16/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5748 -
val_loss: 0.5662
Epoch 17/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5631 -
val_loss: 0.5536
Epoch 18/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5516 -
val_loss: 0.5449
Epoch 19/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5411 -
val_loss: 0.5494
Epoch 20/100
```

```
242/242 [==============================] - 0s 1ms/step - loss: 0.5316 -
val_loss: 0.5256
Epoch 21/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5218 -
val_loss: 0.5111
Epoch 22/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5128 -
val_loss: 0.5090
Epoch 23/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5044 -
val_loss: 0.4914
Epoch 24/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4960 -
val_loss: 0.4745
Epoch 25/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4884 -
val_loss: 0.4765
Epoch 26/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4812 -
val_loss: 0.4611
Epoch 27/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4743 -
val_loss: 0.4683
Epoch 28/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4680 -
val_loss: 0.4481
Epoch 29/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4617 -
val_loss: 0.4433
Epoch 30/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4564 -
val_loss: 0.4453
Epoch 31/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4510 -
val_loss: 0.4486
Epoch 32/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4462 -
val_loss: 0.4260
Epoch 33/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4414 -
val_loss: 0.4290
Epoch 34/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4371 -
val_loss: 0.4143
Epoch 35/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4331 -
val_loss: 0.4202
Epoch 36/100
```

```
242/242 [==============================] - 0s 1ms/step - loss: 0.4293 -
val_loss: 0.4051
Epoch 37/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4259 -
val_loss: 0.4023
Epoch 38/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4226 -
val_loss: 0.4005
Epoch 39/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4195 -
val_loss: 0.3964
Epoch 40/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4166 -
val_loss: 0.4000
Epoch 41/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4138 -
val_loss: 0.3882
Epoch 42/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4117 -
val_loss: 0.3856
Epoch 43/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4091 -
val_loss: 0.3906
Epoch 44/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4073 -
val_loss: 0.3815
Epoch 45/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4051 -
val_loss: 0.3815
Epoch 46/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4032 -
val_loss: 0.3801
Epoch 47/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4013 -
val_loss: 0.3793
Epoch 48/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3997 -
val_loss: 0.3757
Epoch 49/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3979 -
val_loss: 0.3723
Epoch 50/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3965 -
val_loss: 0.3709
Epoch 51/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3948 -
val_loss: 0.3701
Epoch 52/100
```

```
242/242 [==============================] - 0s 1ms/step - loss: 0.3935 -
val_loss: 0.3686
Epoch 53/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3921 -
val_loss: 0.3671
Epoch 54/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3909 -
val_loss: 0.3664
Epoch 55/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3895 -
val_loss: 0.3651
Epoch 56/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3882 -
val_loss: 0.3651
Epoch 57/100
242/242 [==============================] - 0s 2ms/step - loss: 0.3869 -
val_loss: 0.3646
Epoch 58/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3860 -
val_loss: 0.3634
Epoch 59/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3848 -
val_loss: 0.3630
Epoch 60/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3835 -
val_loss: 0.3599
Epoch 61/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3826 -
val_loss: 0.3620
Epoch 62/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3815 -
val_loss: 0.3639
Epoch 63/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3806 -
val_loss: 0.3586
Epoch 64/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3796 -
val_loss: 0.3567
Epoch 65/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3787 -
val_loss: 0.3559
Epoch 66/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3776 -
val_loss: 0.3609
Epoch 67/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3768 -
val_loss: 0.3544
Epoch 68/100
```

```
242/242 [==============================] - 0s 1ms/step - loss: 0.3759 -
val_loss: 0.3575
Epoch 69/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3749 -
val_loss: 0.3545
Epoch 70/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3741 -
val_loss: 0.3557
Epoch 71/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3732 -
val_loss: 0.3523
Epoch 72/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3725 -
val_loss: 0.3541
Epoch 73/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3715 -
val_loss: 0.3516
Epoch 74/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3709 -
val_loss: 0.3506
Epoch 75/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3700 -
val_loss: 0.3554
Epoch 76/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3691 -
val_loss: 0.3601
Epoch 77/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3686 -
val_loss: 0.3503
Epoch 78/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3678 -
val_loss: 0.3484
Epoch 79/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3670 -
val_loss: 0.3489
Epoch 80/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3664 -
val_loss: 0.3530
Epoch 81/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3657 -
val_loss: 0.3517
Epoch 82/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3651 -
val_loss: 0.3513
Epoch 83/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3644 -
val_loss: 0.3491
Epoch 84/100
```

```
242/242 [==============================] - 0s 1ms/step - loss: 0.3636 -
val_loss: 0.3575
Epoch 85/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3630 -
val_loss: 0.3431
Epoch 86/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3626 -
val_loss: 0.3468
Epoch 87/100
242/242 [==============================] - 0s 2ms/step - loss: 0.3618 -
val_loss: 0.3550
Epoch 88/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3614 -
val_loss: 0.3472
Epoch 89/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3607 -
val_loss: 0.3518
Epoch 90/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3601 -
val_loss: 0.3443
Epoch 91/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3595 -
val_loss: 0.3429
Epoch 92/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3592 -
val_loss: 0.3407
Epoch 93/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3583 -
val_loss: 0.3464
Epoch 94/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3579 -
val_loss: 0.3461
Epoch 95/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3572 -
val_loss: 0.3455
Epoch 96/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3568 -
val_loss: 0.3459
Epoch 97/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3563 -
val_loss: 0.3474
Epoch 98/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3560 -
val_loss: 0.3378
Epoch 99/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3553 -
val_loss: 0.3440
Epoch 100/100
```

```
242/242 [==============================] - 0s 1ms/step - loss: 0.3548 -
val_loss: 0.3543
121/121 [==============================] - 0s 793us/step - loss: 0.3551
[CV]  learning_rate=0.00037192261022352417, n_hidden=3, n_neurons=80, total=
34.9s
[CV] learning_rate=0.0008763224455697141, n_hidden=1, n_neurons=47 …
Epoch 1/100
242/242 [==============================] - 0s 2ms/step - loss: 2.3534 -
val_loss: 1.7263
Epoch 2/100
242/242 [==============================] - 0s 1ms/step - loss: 0.9665 -
val_loss: 0.8587
Epoch 3/100
242/242 [==============================] - 0s 1ms/step - loss: 0.7530 -
val_loss: 0.7159
Epoch 4/100
242/242 [==============================] - 0s 1ms/step - loss: 0.7011 -
val_loss: 0.6706
Epoch 5/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6715 -
val_loss: 0.6388
Epoch 6/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6474 -
val_loss: 0.6113
Epoch 7/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6265 -
val_loss: 0.5887
Epoch 8/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6076 -
val_loss: 0.5725
Epoch 9/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5901 -
val_loss: 0.5671
Epoch 10/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5740 -
val_loss: 0.5424
Epoch 11/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5596 -
val_loss: 0.5278
Epoch 12/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5465 -
val_loss: 0.5211
Epoch 13/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5343 -
val_loss: 0.5037
Epoch 14/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5233 -
val_loss: 0.4960
```

```
Epoch 15/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5133 -
val_loss: 0.4914
Epoch 16/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5039 -
val_loss: 0.4789
Epoch 17/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4957 -
val_loss: 0.4763
Epoch 18/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4879 -
val_loss: 0.4646
Epoch 19/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4809 -
val_loss: 0.4627
Epoch 20/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4746 -
val_loss: 0.4551
Epoch 21/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4689 -
val_loss: 0.4578
Epoch 22/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4636 -
val_loss: 0.4501
Epoch 23/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4588 -
val_loss: 0.4457
Epoch 24/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4543 -
val_loss: 0.4383
Epoch 25/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4506 -
val_loss: 0.4419
Epoch 26/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4468 -
val_loss: 0.4435
Epoch 27/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4435 -
val_loss: 0.4356
Epoch 28/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4404 -
val_loss: 0.4393
Epoch 29/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4374 -
val_loss: 0.4341
Epoch 30/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4348 -
val_loss: 0.4290
```

```
Epoch 31/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4322 -
val_loss: 0.4222
Epoch 32/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4298 -
val_loss: 0.4217
Epoch 33/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4276 -
val_loss: 0.4261
Epoch 34/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4254 -
val_loss: 0.4312
Epoch 35/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4234 -
val_loss: 0.4219
Epoch 36/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4214 -
val_loss: 0.4262
Epoch 37/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4196 -
val_loss: 0.4137
Epoch 38/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4178 -
val_loss: 0.4255
Epoch 39/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4161 -
val_loss: 0.4136
Epoch 40/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4144 -
val_loss: 0.4214
Epoch 41/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4128 -
val_loss: 0.4199
Epoch 42/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4113 -
val_loss: 0.4110
Epoch 43/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4098 -
val_loss: 0.4056
Epoch 44/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4082 -
val_loss: 0.4185
Epoch 45/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4070 -
val_loss: 0.4069
Epoch 46/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4055 -
val_loss: 0.4047
```

```
Epoch 47/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4042 -
val_loss: 0.4145
Epoch 48/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4029 -
val_loss: 0.4071
Epoch 49/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4017 -
val_loss: 0.4045
Epoch 50/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4004 -
val_loss: 0.4106
Epoch 51/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3992 -
val_loss: 0.4045
Epoch 52/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3982 -
val_loss: 0.4016
Epoch 53/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3970 -
val_loss: 0.4060
Epoch 54/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3959 -
val_loss: 0.3909
Epoch 55/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3947 -
val_loss: 0.4049
Epoch 56/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3939 -
val_loss: 0.3950
Epoch 57/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3928 -
val_loss: 0.3980
Epoch 58/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3918 -
val_loss: 0.4034
Epoch 59/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3909 -
val_loss: 0.3910
Epoch 60/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3899 -
val_loss: 0.3907
Epoch 61/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3892 -
val_loss: 0.3970
Epoch 62/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3882 -
val_loss: 0.4017
```

```
Epoch 63/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3874 -
val_loss: 0.3941
Epoch 64/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3866 -
val_loss: 0.4017
Epoch 65/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3858 -
val_loss: 0.3987
Epoch 66/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3852 -
val_loss: 0.3860
Epoch 67/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3842 -
val_loss: 0.3922
Epoch 68/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3835 -
val_loss: 0.3874
Epoch 69/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3827 -
val_loss: 0.3902
Epoch 70/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3821 -
val_loss: 0.3876
Epoch 71/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3813 -
val_loss: 0.3953
Epoch 72/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3806 -
val_loss: 0.3857
Epoch 73/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3801 -
val_loss: 0.3787
Epoch 74/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3793 -
val_loss: 0.3875
Epoch 75/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3787 -
val_loss: 0.3882
Epoch 76/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3780 -
val_loss: 0.3865
Epoch 77/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3776 -
val_loss: 0.3733
Epoch 78/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3768 -
val_loss: 0.3836
```

```
Epoch 79/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3762 -
val_loss: 0.3826
Epoch 80/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3756 -
val_loss: 0.3812
Epoch 81/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3750 -
val_loss: 0.3861
Epoch 82/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3746 -
val_loss: 0.3804
Epoch 83/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3740 -
val_loss: 0.3749
Epoch 84/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3734 -
val_loss: 0.3757
Epoch 85/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3728 -
val_loss: 0.3754
Epoch 86/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3721 -
val_loss: 0.3793
Epoch 87/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3720 -
val_loss: 0.3647
Epoch 88/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3714 -
val_loss: 0.3656
Epoch 89/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3708 -
val_loss: 0.3671
Epoch 90/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3703 -
val_loss: 0.3779
Epoch 91/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3700 -
val_loss: 0.3642
Epoch 92/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3695 -
val_loss: 0.3627
Epoch 93/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3690 -
val_loss: 0.3722
Epoch 94/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3687 -
val_loss: 0.3641
```

```
Epoch 95/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3682 -
val_loss: 0.3619
Epoch 96/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3676 -
val_loss: 0.3613
Epoch 97/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3672 -
val_loss: 0.3584
Epoch 98/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3669 -
val_loss: 0.3593
Epoch 99/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3662 -
val_loss: 0.3744
Epoch 100/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3661 -
val_loss: 0.3576
121/121 [==============================] - 0s 911us/step - loss: 0.3824
[CV]  learning_rate=0.0008763224455697141, n_hidden=1, n_neurons=47, total=
31.3s
[CV] learning_rate=0.0008763224455697141, n_hidden=1, n_neurons=47 …
Epoch 1/100
242/242 [==============================] - 0s 1ms/step - loss: 3.1701 -
val_loss: 10.4915
Epoch 2/100
242/242 [==============================] - 0s 1ms/step - loss: 1.1445 -
val_loss: 8.4356
Epoch 3/100
242/242 [==============================] - 0s 1ms/step - loss: 0.8441 -
val_loss: 4.5418
Epoch 4/100
242/242 [==============================] - 0s 1ms/step - loss: 0.7477 -
val_loss: 2.1160
Epoch 5/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6982 -
val_loss: 0.9668
Epoch 6/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6637 -
val_loss: 0.6130
Epoch 7/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6359 -
val_loss: 0.7181
Epoch 8/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6123 -
val_loss: 1.0097
Epoch 9/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5918 -
```

```
val_loss: 1.3441
Epoch 10/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5738 -
val_loss: 1.6915
Epoch 11/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5578 -
val_loss: 2.0120
Epoch 12/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5436 -
val_loss: 2.3420
Epoch 13/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5313 -
val_loss: 2.5808
Epoch 14/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5203 -
val_loss: 2.6797
Epoch 15/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5105 -
val_loss: 2.8334
Epoch 16/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5017 -
val_loss: 2.8812
121/121 [==============================] - 0s 827us/step - loss: 0.5691
[CV]  learning_rate=0.0008763224455697141, n_hidden=1, n_neurons=47, total=
5.5s
[CV] learning_rate=0.0008763224455697141, n_hidden=1, n_neurons=47 …
Epoch 1/100
242/242 [==============================] - 0s 2ms/step - loss: 2.8964 -
val_loss: 1.3028
Epoch 2/100
242/242 [==============================] - 0s 1ms/step - loss: 0.9874 -
val_loss: 0.8500
Epoch 3/100
242/242 [==============================] - 0s 1ms/step - loss: 0.7410 -
val_loss: 0.6924
Epoch 4/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6737 -
val_loss: 0.6281
Epoch 5/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6375 -
val_loss: 0.6377
Epoch 6/100
242/242 [==============================] - 0s 1ms/step - loss: 0.6098 -
val_loss: 0.5742
Epoch 7/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5873 -
val_loss: 0.5536
Epoch 8/100
```

```
242/242 [==============================] - 0s 1ms/step - loss: 0.5677 -
val_loss: 0.5556
Epoch 9/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5527 -
val_loss: 0.5260
Epoch 10/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5375 -
val_loss: 0.5178
Epoch 11/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5241 -
val_loss: 0.4904
Epoch 12/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5125 -
val_loss: 0.4839
Epoch 13/100
242/242 [==============================] - 0s 1ms/step - loss: 0.5018 -
val_loss: 0.4790
Epoch 14/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4923 -
val_loss: 0.4765
Epoch 15/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4837 -
val_loss: 0.4634
Epoch 16/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4762 -
val_loss: 0.4485
Epoch 17/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4693 -
val_loss: 0.4413
Epoch 18/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4634 -
val_loss: 0.4372
Epoch 19/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4579 -
val_loss: 0.4329
Epoch 20/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4529 -
val_loss: 0.4328
Epoch 21/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4484 -
val_loss: 0.4318
Epoch 22/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4444 -
val_loss: 0.4201
Epoch 23/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4407 -
val_loss: 0.4198
Epoch 24/100
```

```
242/242 [==============================] - 0s 1ms/step - loss: 0.4371 -
val_loss: 0.4364
Epoch 25/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4342 -
val_loss: 0.4068
Epoch 26/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4312 -
val_loss: 0.4062
Epoch 27/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4284 -
val_loss: 0.4216
Epoch 28/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4262 -
val_loss: 0.4045
Epoch 29/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4234 -
val_loss: 0.4330
Epoch 30/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4218 -
val_loss: 0.4084
Epoch 31/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4196 -
val_loss: 0.3998
Epoch 32/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4177 -
val_loss: 0.4304
Epoch 33/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4160 -
val_loss: 0.4137
Epoch 34/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4145 -
val_loss: 0.4342
Epoch 35/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4135 -
val_loss: 0.3982
Epoch 36/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4117 -
val_loss: 0.3970
Epoch 37/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4106 -
val_loss: 0.3829
Epoch 38/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4090 -
val_loss: 0.4255
Epoch 39/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4082 -
val_loss: 0.3857
Epoch 40/100
```

```
242/242 [==============================] - 0s 1ms/step - loss: 0.4068 -
val_loss: 0.3945
Epoch 41/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4055 -
val_loss: 0.4352
Epoch 42/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4050 -
val_loss: 0.4157
Epoch 43/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4037 -
val_loss: 0.3750
Epoch 44/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4030 -
val_loss: 0.4069
Epoch 45/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4021 -
val_loss: 0.3918
Epoch 46/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4012 -
val_loss: 0.4089
Epoch 47/100
242/242 [==============================] - 0s 1ms/step - loss: 0.4005 -
val_loss: 0.3709
Epoch 48/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3996 -
val_loss: 0.3759
Epoch 49/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3985 -
val_loss: 0.4202
Epoch 50/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3983 -
val_loss: 0.4091
Epoch 51/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3974 -
val_loss: 0.4066
Epoch 52/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3967 -
val_loss: 0.4156
Epoch 53/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3965 -
val_loss: 0.3709
Epoch 54/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3956 -
val_loss: 0.3675
Epoch 55/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3948 -
val_loss: 0.3697
Epoch 56/100
```

```
242/242 [==============================] - 0s 1ms/step - loss: 0.3940 -
val_loss: 0.3795
Epoch 57/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3933 -
val_loss: 0.3917
Epoch 58/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3931 -
val_loss: 0.3966
Epoch 59/100
242/242 [==============================] - 0s 1ms/step - loss: 0.3925 -
val_loss: 0.3901
Epoch 60/100
191/242 [======================>…] - ETA: 0s - loss: 0.4023
```

[ ]:
```
# describe best model
rnd_search_cv.best_params_
```

[ ]:
```
# best validation score
rnd_search_cv.best_score_
```

[ ]:
```
-0.35952892616378346
```

[ ]:
```
rnd_search_cv.best_estimator_
```

[ ]:
```
<tensorflow.python.keras.wrappers.scikit_learn.KerasRegressor at 0x7ff384301518>
```

[ ]:
```
rnd_search_cv.score(X_test, y_test)
# Testing data showes lower loss than in the untuned model.
```

```
5160/5160 [==============================] - 0s 15us/sample - loss: 0.3065
```

[ ]:
```
-0.30652404945026074
```

[ ]:
```
model = rnd_search_cv.best_estimator_.model
model
```

[ ]:
```
<tensorflow.python.keras.engine.sequential.Sequential at 0x7ff350924668>
```

[ ]:
```
model.evaluate(X_test, y_test)
```

```
5160/5160 [==============================] - 0s 15us/sample - loss: 0.3065
```

[ ]:
```
0.30652404945026074
```