Co-funded by the
Erasmus+ Programme
of the European Union

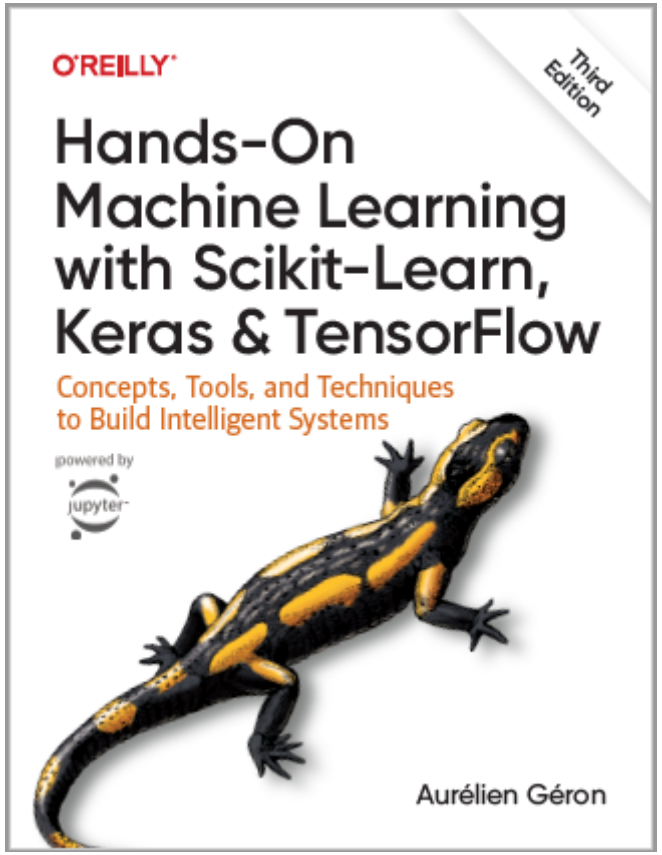
DeCAIR

Artificial Neural Networks with Keras

Prof. Gheith Abandah

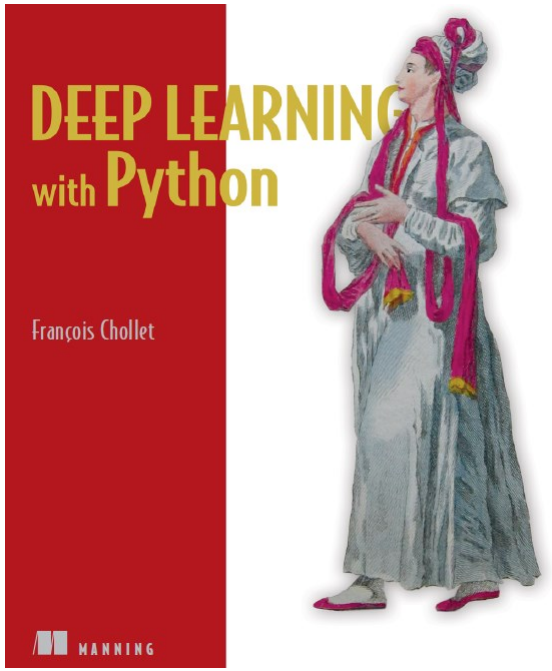
Reference

- Chapter 10: **Introduction to Artificial Neural Networks with Keras**



- Aurélien Géron, **Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow**, O'Reilly, 3rd Edition, 2022
  - Material: https://github.com/ageron/handson-ml3

Reference



- **Deep Learning with Python**, by François Chollet, Manning Pub. 2018

Outline

1. Introduction
2. Keras API Styles
3. TensorFlow Keras
4. Image Classifier Using the Sequential Model
5. Regression Using the Sequential Model
6. Using the Functional API
7. Using Callbacks
8. Visualization Using TensorBoard
9. Fine-Tuning Neural Network Hyperparameters
10. Tutorials
11. Exercise

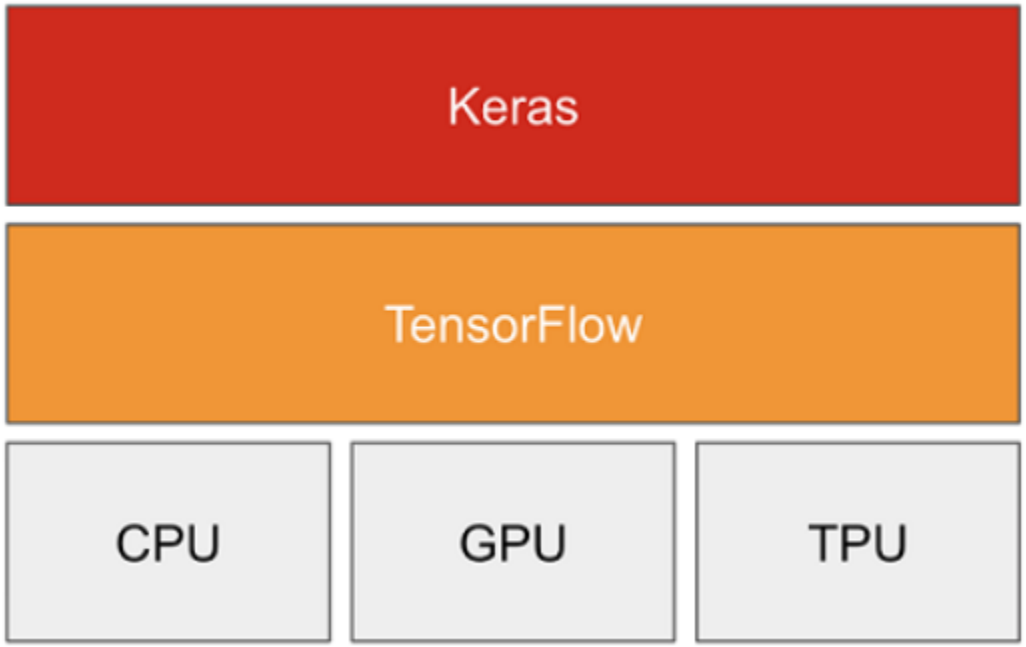
Introduction

- YouTube Video: *Keras Explained* from Siraj Raval

https://youtu.be/j_pJmXJwMLA

1. Introduction

- **Keras** is a high-level API to build and train deep learning models.



1. Introduction – Advantages

- **User friendly**: Keras has a simple, consistent interface optimized for common use cases. It provides clear and actionable feedback for user errors.
- **Modular and composable**: Keras models are made by connecting configurable building blocks together, with few restrictions.
- **Easy to extend**: Write custom building blocks to express new ideas for research. Create new layers, loss functions, and develop state-of-the-art models.

Outline

1. Introduction
2. Keras API Styles
3. TensorFlow Keras
4. Image Classifier Using the Sequential Model
5. Regression Using the Sequential Model
6. Using the Functional API
7. Using Callbacks
8. Visualization Using TensorBoard
9. Fine-Tuning Neural Network Hyperparameters
10. Tutorials
11. Exercise

2. Keras API Styles

1. The Sequential Model

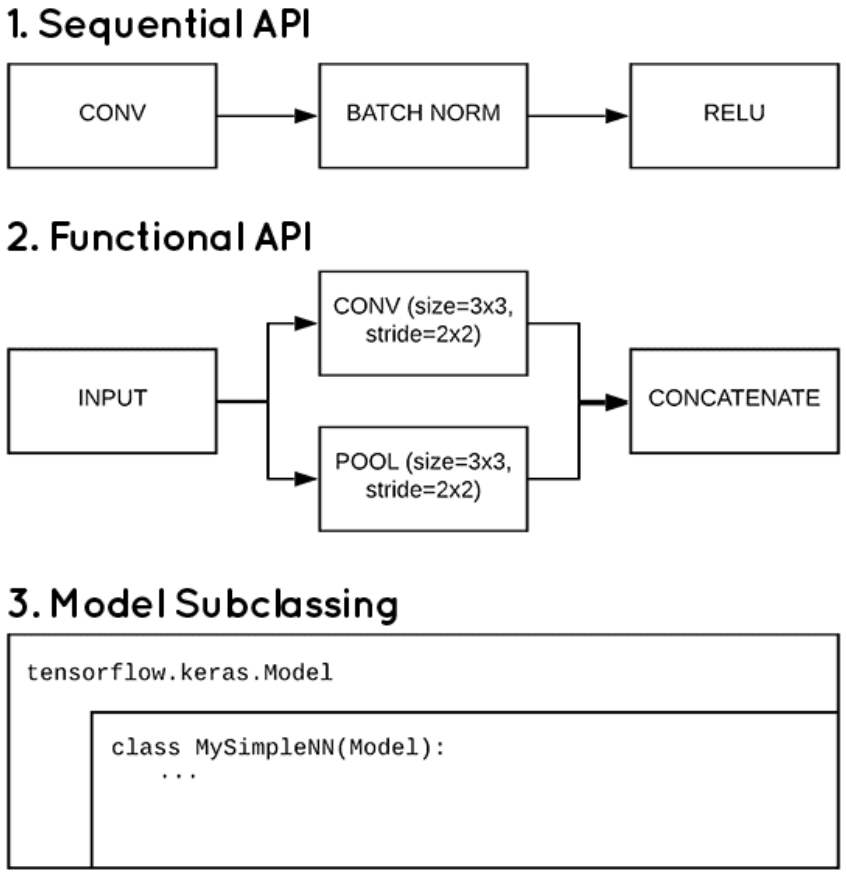
- Dead simple
- Only for single-input, single-output, sequential layer stacks
- Good for 70+% of use cases

2. The Functional API

- Like playing with Lego bricks
- Multi-input, multi-output, arbitrary static graph topologies
- Good for 95% of use cases

3. Model Subclassing

- Maximum flexibility
- Larger potential error surface

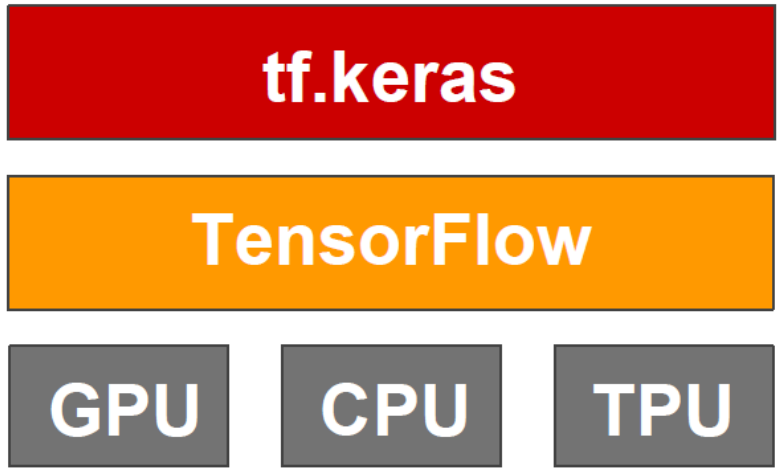


Outline

1. Introduction
2. Keras API Styles
3. TensorFlow Keras
4. Image Classifier Using the Sequential Model
5. Regression Using the Sequential Model
6. Using the Functional API
7. Using Callbacks
8. Visualization Using TensorBoard
9. Fine-Tuning Neural Network Hyperparameters
10. Tutorials
11. Exercise

3. TensorFlow Keras

- Keras is the official high-level API of TensorFlow
- tensorflow.keras (tf.keras) module
- Part of core TensorFlow since v1.4
- Full Keras API
- With useful extra features such as **tf.data**



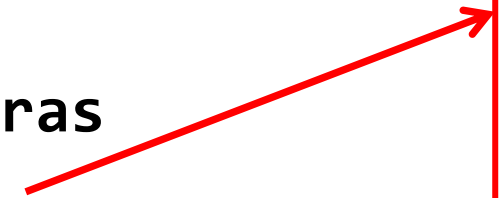
3. TensorFlow Keras

- To install TensorFlow

```
$ pip install --upgrade tensorflow
```

- To import Keras from TensorFlow

```
>>> import tensorflow as tf
>>> from tensorflow import keras
>>> from keras import layers
>>> tf.__version__
'2.9.2'
>>> keras.__version__
'2.9.0'
```



- Dense
- Activations
- Dropout
- Conv1D, 2D, 3D
- Polling
- RNN, LSTM, GRU
- …

Outline

1. Introduction
2. Keras API Styles
3. TensorFlow Keras
4. Image Classifier Using the Sequential Model
5. Regression Using the Sequential Model
6. Using the Functional API
7. Using Callbacks
8. Visualization Using TensorBoard
9. Fine-Tuning Neural Network Hyperparameters
10. Tutorials
11. Exercise

4. Image Classifier Using the Sequential Model

- **Fashion MNIST** is like MNIST (70,000 grayscale images of 28×28 pixels each, with 10 classes).

4. Fashion MNIST

1. **Get and prepare the dataset**.
2. **Build sequential model** of layers that maps your inputs to your targets.
3. **Compile the model and configure the learning process** by choosing a loss function, an optimizer, and some metrics to monitor.
4. **Train the model** by calling the `fit()` method of your model.
5. **Evaluate and use** the model.

4.1 Get and Prepare the Dataset

```
import tensorflow as tf
from tensorflow import keras

# Get the Fashion MNIST
fashion_mnist = keras.datasets.fashion_mnist.load_data()
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist

# Prepare the data val (5000), train (55000), test (10000)
X_train, y_train = X_train_full[:-5000], y_train_full[:-5000]
X_valid, y_valid = X_train_full[-5000:], y_train_full[-5000:]

X_train, X_valid, X_test = X_train / 255., X_valid / 255., X_test / 255.
```

```
>>> X_train_full.shape
(60000, 28, 28)
>>> X_train_full.dtype
dtype('uint8')
```

4.2 Build the Model

```
model = keras.Sequential()
model.add(layers.Input(shape=[28, 28]))
model.add(layers.Flatten())
model.add(layers.Dense(300, activation="relu"))
model.add(layers.Dense(100, activation="relu"))
model.add(layers.Dense(10, activation="softmax"))
```

The default is no activation function, *i.e.*, linear layer.

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

4.2 Build the Model

- You can check the model.
- You can get the model's list of layers.

```
>>> model.layers
[<keras.layers.core.flatten.Flatten at 0x7fa1dea02250>,
 <keras.layers.core.dense.Dense at 0x7fa1c8f42520>,
 <keras.layers.core.dense.Dense at 0x7fa188be7ac0>,
 <keras.layers.core.dense.Dense at 0x7fa188be7fa0>]
>>> hidden1 = model.layers[1]
>>> hidden1.name
'dense'
>>> model.get_layer('dense') is hidden1
True
```

```
>>> model.summary()
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
_________________________________________________________________
```

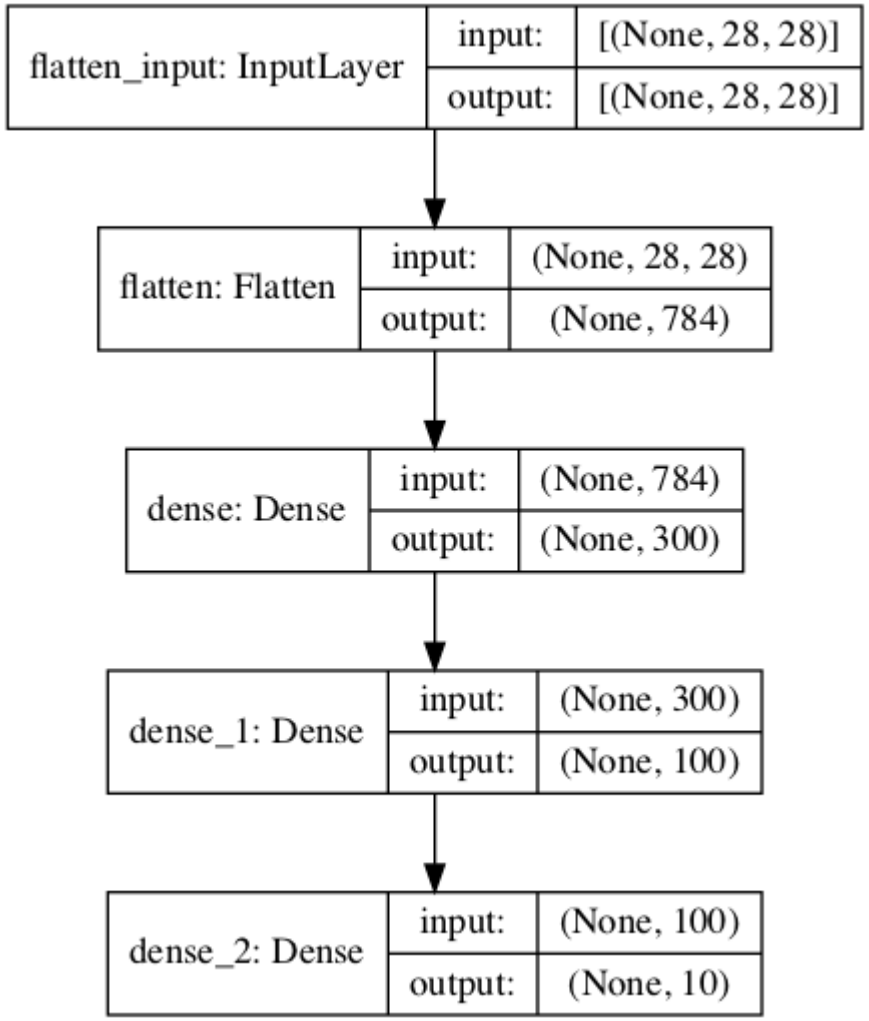
4.2 Build the Model

- All the parameters of a layer can be accessed using **`get_weights()`**.

```
>>> weights, biases = hidden1.get_weights()
>>> weights
array([[ 0.02448617, -0.00877795, -0.02189048, ...,  0.03859074, -0.06889391],
       [ 0.00476504, -0.03105379, -0.0586676 , ..., -0.02763776, -0.04165364],
       ...,
       [ 0.07061854, -0.06960931,  0.07038955, ..., 0.00034875,  0.02878492],
       [-0.06022581,  0.01577859, -0.02585464, ..., 0.00272203, -0.06793761]],
       dtype=float32)
>>> weights.shape
(784, 300)
>>> biases
array([0., 0., 0., 0., 0., 0., 0., 0., 0., ...,  0., 0., 0.], dtype=float32)
>>> biases.shape
(300,)
```

4.2 Build the Model

```
# Plot the model
keras.utils.plot_model(
        model,
        "my_model.png",
        show_shapes=True)
```



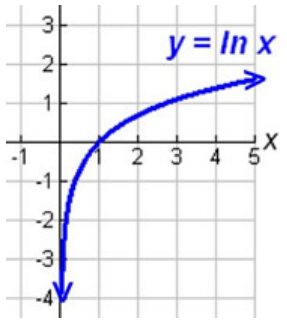
4.3 Compile the Model

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])

# For sparse labels (0-9):
loss = "sparse_categorical_crossentropy"
# For one-hot labels:
loss = "categorical_crossentropy"
# For binary labels:
loss = "binary_crossentropy"
# For regression:
loss = "mean_squared_error"
```

Stochastic Gradient Descent

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$



4.4 Train the Model

```
# Train the model
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```

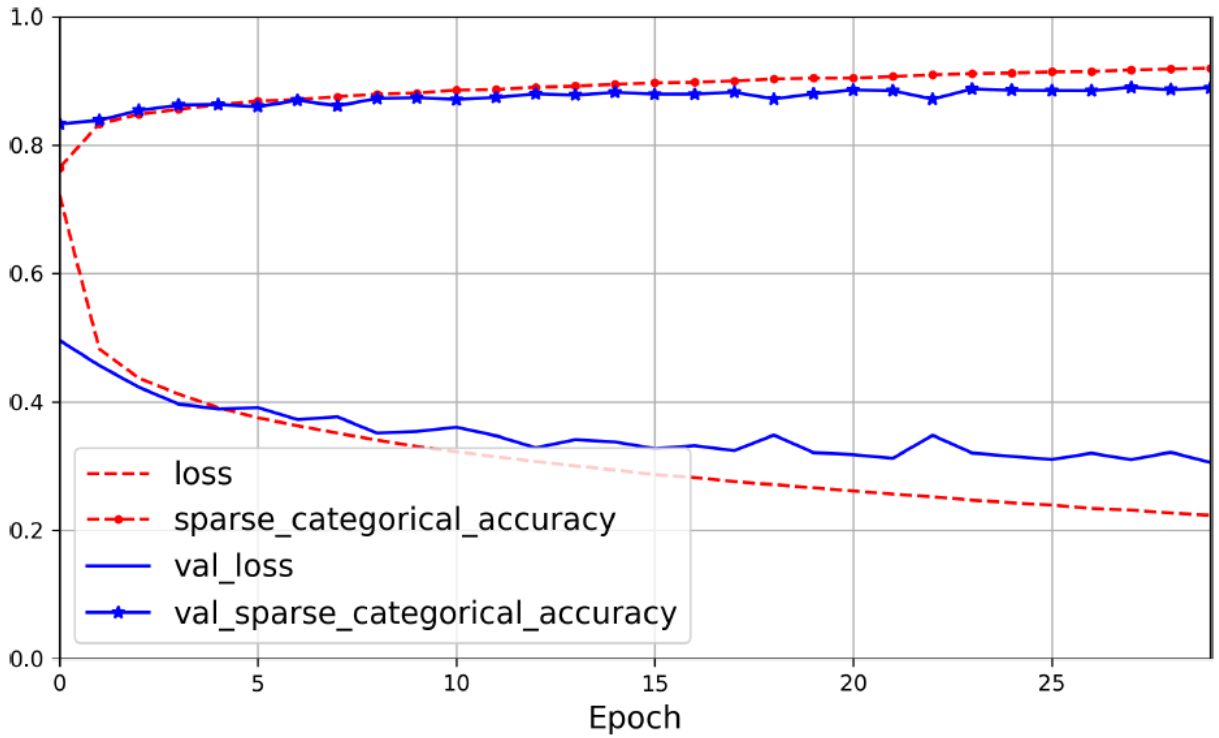
Alternative:
`validation_split=0.08`

```
Epoch 1/30
1719/1719 [==============================] - 2s 989us/step
- loss: 0.7220 - sparse_categorical_accuracy: 0.7649
- val_loss: 0.4959 - val_sparse_categorical_accuracy: 0.8332
Epoch 2/30
1719/1719 [==============================] - 2s 964us/step
- loss: 0.4825 - sparse_categorical_accuracy: 0.8332
- val_loss: 0.4567 - val_sparse_categorical_accuracy: 0.8384
[...]
Epoch 30/30
1719/1719 [==============================] - 2s 963us/step
- loss: 0.2235 - sparse_categorical_accuracy: 0.9200
- val_loss: 0.3056 - val_sparse_categorical_accuracy: 0.8894
```

Default batch size is 32

4.4 Train the Model

```
import matplotlib.pyplot as plt
import pandas as pd
pd.DataFrame(history.history).plot(
    figsize=(8, 5), xlim=[0, 29], ylim=[0, 1], grid=True, xlabel="Epoch",
    style=["r--", "r--.", "b-", "b-*"])
plt.show()
```



4.5 Evaluate and Use the Model

```
model.evaluate(X_test, y_test)
313/313 [==============================] - 0s 626us/step
- loss: 0.3243 - sparse_categorical_accuracy: 0.8864
[0.3243169784545898 4, 0.8863999843597412]

X_new = X_test[:3]
y_proba = model.predict(X_new)
y_proba.round(2)
array([[0. , 0. , 0.  , 0. , 0.  , 0.01, 0. , 0.02, 0. , 0.97],
       [0. , 0. , 0.99, 0. , 0.01, 0.  , 0. , 0.  , 0. , 0.  ],
       [0. , 1. , 0.  , 0. , 0.  , 0.  , 0. , 0.  , 0. , 0.  ]],
      dtype=float32)

model.predict_classes(X_new)
array([9, 2, 1])
```

```
>>> import numpy as np
>>> y_pred = y_proba.argmax(axis=-1)
>>> y_pred
array([9, 2, 1])
```

Outline

1. Introduction
2. Keras API Styles
3. TensorFlow Keras
4. Image Classifier Using the Sequential Model
5. Regression Using the Sequential Model
6. Using the Functional API
7. Using Callbacks
8. Visualization Using TensorBoard
9. Fine-Tuning Neural Network Hyperparameters
10. Tutorials
11. Exercise

5. Regression Using the Sequential Model

- Solve the **California housing** problem using a regression neural network.
- Scikit-Learn has `fetch_california_housing()` function to load the data
- This dataset contains **only numerical features** and there are **no missing values**.

5.1 Get and Prepare the Dataset

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split

housing = fetch_california_housing()

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)
```

The default is 75% : 25%

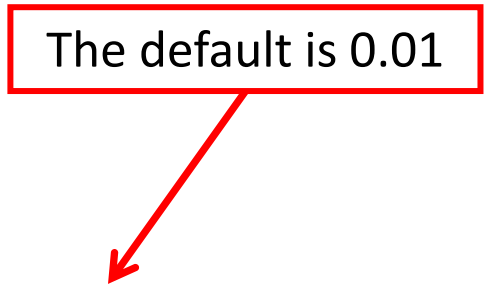
5.2 Build the Model

Same function as **StandardScaler**

```
# Construct a normalization layer
norm_layer = layers.Normalization(input_shape=X_train.shape[1:])

# Building by passing a list of layers when creating
#  the Sequential model
model = keras.Sequential([
    norm_layer,
    layers.Dense(50, activation="relu"),
    layers.Dense(50, activation="relu"),
    layers.Dense(50, activation="relu"),
    layers.Dense(1)
])
```

5.3 Compile the Model

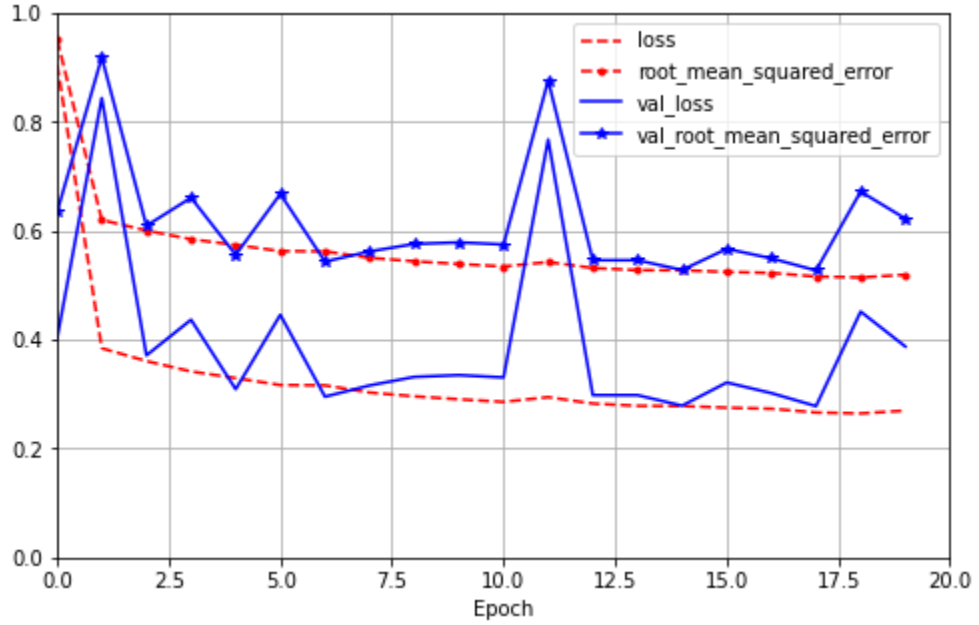



```
# Compile with creating an optimizer object
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
model.compile(loss="mse", optimizer=optimizer,
              metrics=["RootMeanSquaredError"])

# Fit the normalization layer on the training set
norm_layer.adapt(X_train)
```

5.4 Train and Evaluate the Model

```
history = model.fit(X_train, y_train, epochs=20,
        validation_data=(X_valid, y_valid))
```



5.4 Train and Evaluate the Model

```
mse_test, rmse_test = model.evaluate(X_test, y_test)
162/162 [==============================] - 0s 2ms/step - loss: 0.2806 -
root_mean_squared_error: 0.5297
```

```
X_new = X_test[:3]
y_pred = model.predict(X_new)
print(y_pred)
[[0.4579417]
 [1.2503718]
 [5.1537175]]
```

5.5 Save and Restore the Model

- After training a model save it to a file.

```
model.save("my_keras_model", save_format="tf")
```

- In the production program, load the trained model.

```
model = keras.models.load_model("my_keras_model")
y_pred = model.predict(X_new)
```

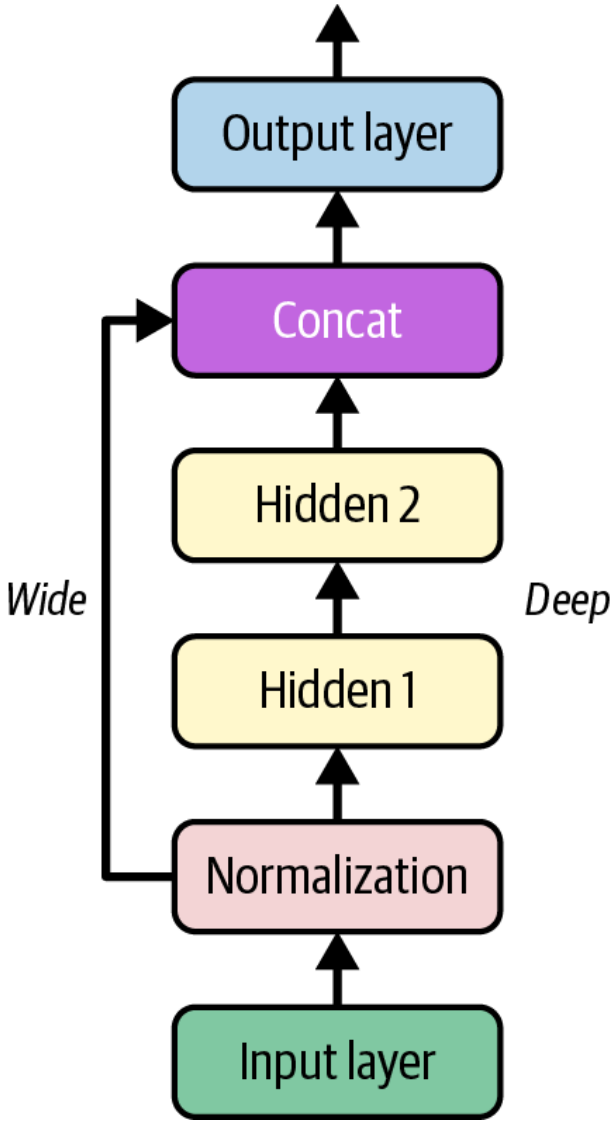
Outline

1. Introduction
2. Keras API Styles
3. TensorFlow Keras
4. Image Classifier Using the Sequential Model
5. Regression Using the Sequential Model
6. Using the Functional API
7. Using Callbacks
8. Visualization Using TensorBoard
9. Fine-Tuning Neural Network Hyperparameters
10. Tutorials
11. Exercise

6. Using the Functional API

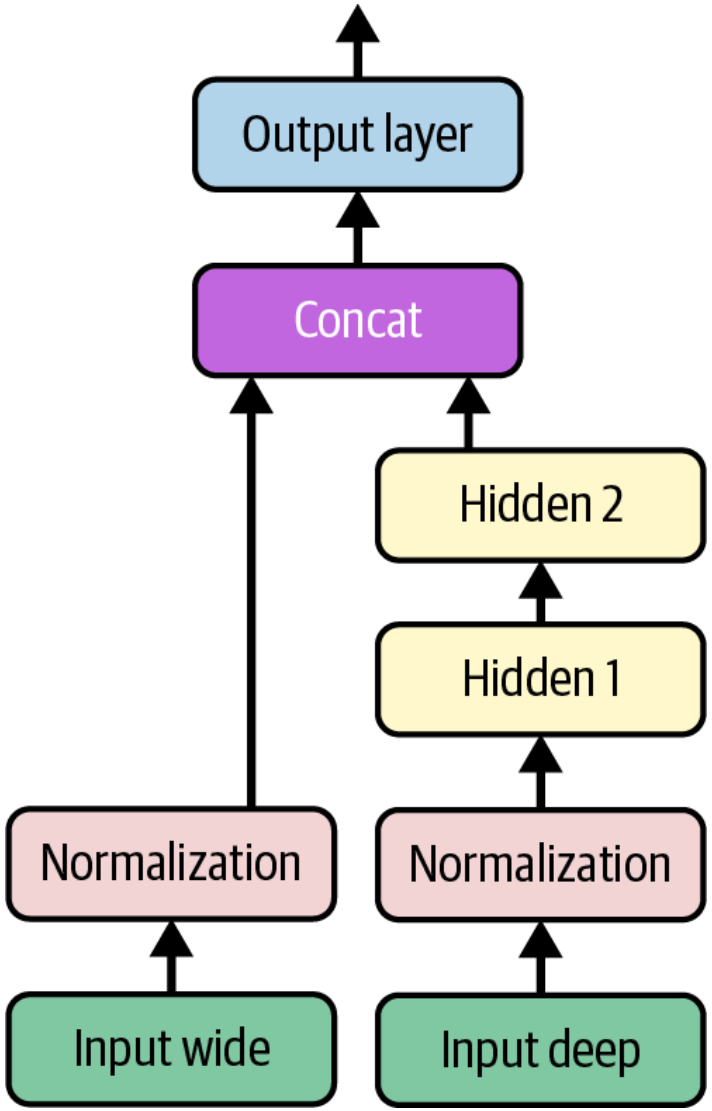
- Keras functional API can be used to build arbitrary **static graph topologies**.
- Create a layer and **call it like a function**, passing it the input.
- **Examples**

1. **Wide and deep** network that learns both deep patterns (using the deep path) and simple rules (through the short path).



6. Using the Functional API

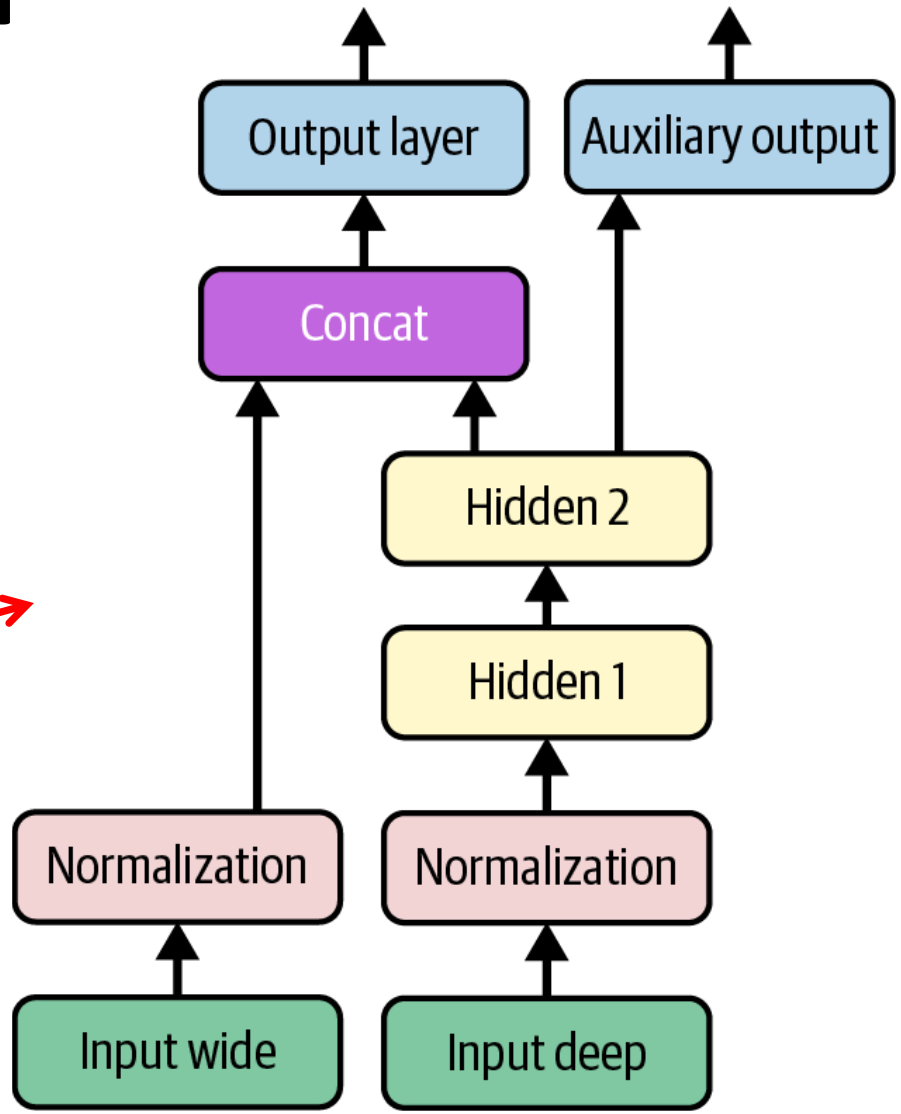
2. **Multi-input**: You can send a subset of the features through the wide path, and a different subset (possibly overlapping) through the deep path.



6. Using the Functional API

3. Multiple Outputs

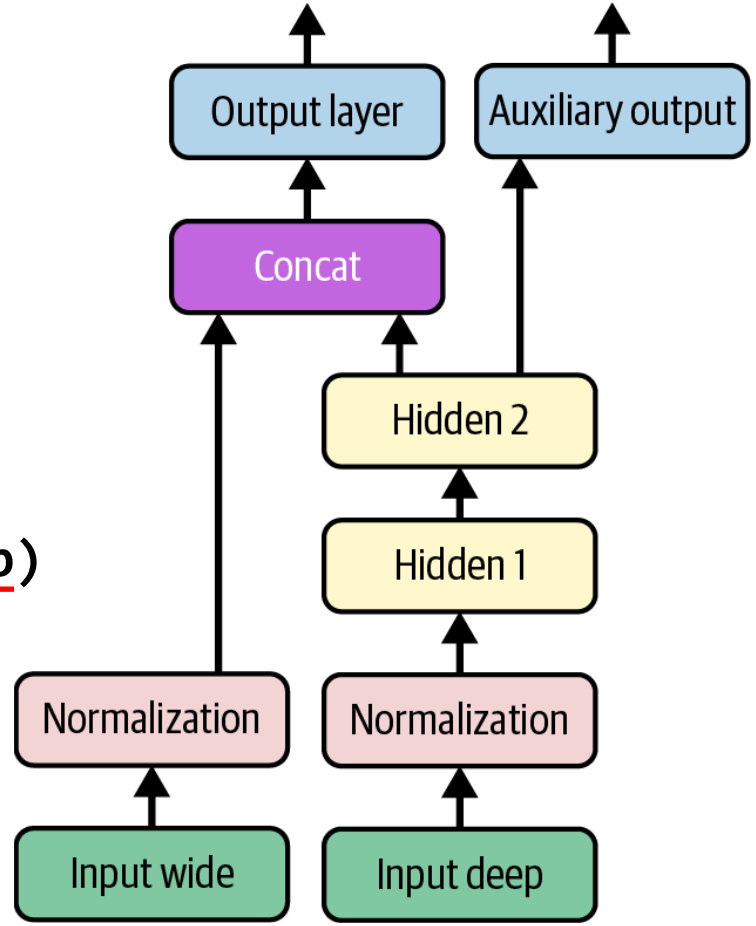
- To **locate and classify** the main object in a picture.
- **Multiple independent tasks** to perform based on the same data.
- **Regularization technique** (to ensure that the deep network learns something useful on its own).



6.1 Auxiliary Output for Regularization

```
# Build the model
input_wide = layers.Input(shape=[5]) # features 0 to 4
input_deep = layers.Input(shape=[6]) # features 2 to 7
norm_layer_wide = layers.Normalization()
norm_layer_deep = layers.Normalization()
norm_wide = norm_layer_wide(input_wide)
norm_deep = norm_layer_deep(input_deep)
hidden1 = layers.Dense(30, activation="relu")(norm_deep)
hidden2 = layers.Dense(30, activation="relu")(hidden1)
concat = layers.concatenate([norm_wide, hidden2])
output = layers.Dense(1)(concat)
aux_output = layers.Dense(1)(hidden2)

model = keras.Model(inputs=[input_wide, input_deep],
                    outputs=[output, aux_output])
```



6.1 Auxiliary Output for Regularization

```
# Split the input
X_train_wide, X_train_deep = X_train[:, :5], X_train[:, 2:]
X_valid_wide, X_valid_deep = X_valid[:, :5], X_valid[:, 2:]
X_test_wide,  X_test_deep  = X_test[:, :5],  X_test[:, 2:]

# Take some test samples
X_new_wide, X_new_deep = X_test_wide[:3], X_test_deep[:3]
```

6.1 Auxiliary Output for Regularization

```
# Compile using adam optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
model.compile(loss=("mse", "mse"), loss_weights=(0.9, 0.1),
              optimizer=optimizer, metrics=["RootMeanSquaredError"])
```

6.1 Auxiliary Output for Regularization

```
# Train, evaluate, and predict
norm_layer_wide.adapt(X_train_wide)
norm_layer_deep.adapt(X_train_deep)
history = model.fit(
    (X_train_wide, X_train_deep), (y_train, y_train), epochs=20,
    validation_data=((X_valid_wide, X_valid_deep), (y_valid, y_valid))
)
weighted_sum_of_losses, main_loss, aux_loss, main_rmse, aux_rmse =
    model.evaluate((X_test_wide, X_test_deep), (y_test, y_test))

y_pred_main, y_pred_aux = model.predict([X_new_wide, X_new_deep])
```

Outline

1. Introduction
2. Keras API Styles
3. TensorFlow Keras
4. Image Classifier Using the Sequential Model
5. Regression Using the Sequential Model
6. Using the Functional API
7. Using Callbacks
8. Visualization Using TensorBoard
9. Fine-Tuning Neural Network Hyperparameters
10. Tutorials
11. Exercise

7. Using Callbacks

- The `fit()` method accepts a `callbacks` argument that lets you specify a list of objects that Keras will call during training
  - at the start and end of **training**
  - at the start and end of each **epoch**
  - before and after processing each **batch**
- There are many callbacks available in the `keras.callbacks` package. See

https://keras.io/callbacks/

7.1 Saving Best Model

- **Save your best model** when its performance on the validation set is the best so far.

```
checkpoint_cb = keras.callbacks.ModelCheckpoint(
        "my_checkpoints", save_best_only=True)

history = model.fit(…, epochs=10, callbacks=[checkpoint_cb])

# rollback to best model
model = keras.models.load_model("my_checkpoints")
mse_test = model.evaluate(X_test, y_test)
```

7.2 Early Stopping

- Interrupt training when there is no progress on the validation set for a number of epochs (defined by the **patience** argument)
- Optionally roll back to the best model.

```
early_stopping_cb = keras.callbacks.EarlyStopping(
        patience=10, restore_best_weights=True)

history = model.fit(…, epochs=100,
                    callbacks=[checkpoint_cb, early_stopping_cb])
```

Outline

1. Introduction
2. Keras API Styles
3. TensorFlow Keras
4. Image Classifier Using the Sequential Model
5. Regression Using the Sequential Model
6. Using the Functional API
7. Using Callbacks
8. Visualization Using TensorBoard
9. Fine-Tuning Neural Network Hyperparameters
10. Tutorials
11. Exercise

8. Visualization Using TensorBoard

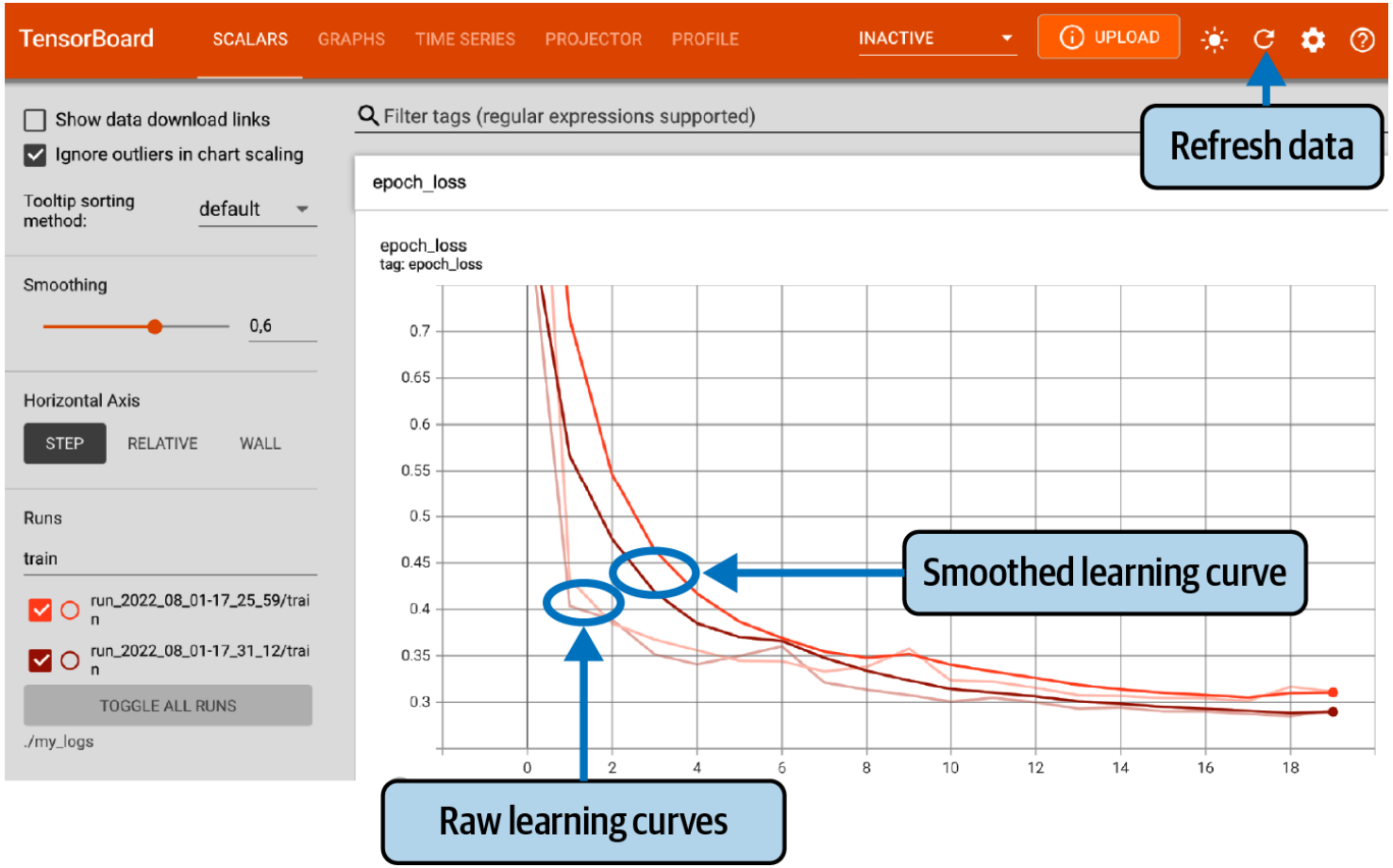
- TensorBoard is a great **interactive visualization tool** that comes with TensorFlow.
- Use it using its callback

```
tensorboard_cb =
        keras.callbacks.TensorBoard(run_logdir)
history = model.fit(…, callbacks=[tensorboard_cb])
```

- Start TensorBoard server

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

8. Open http://localhost:6006



Outline

1. Introduction
2. Keras API Styles
3. TensorFlow Keras
4. Image Classifier Using the Sequential Model
5. Regression Using the Sequential Model
6. Using the Functional API
7. Using Callbacks
8. Visualization Using TensorBoard
9. Fine-Tuning Neural Network Hyperparameters
10. Tutorials
11. Exercise

9. Fine-Tuning Neural Network Hyperparameters

- The **flexibility** of neural networks makes them **hard to tune**.
- **Important Hyperparameters**
  - Number of hidden layers
  - Number of neurons per hidden layer
  - Learning rate, optimizer, batch size, activation function, number of iterations
- You can use Scikit-Learn `GridSearchCV` or `RandomizedSearchCV` after wrapping your NN with `KerasRegressor` or `KerasClassifier`.
- Also, Keras has `keras_tuner` that can search for best hyperparameters using a function that builds the NN.

Extra Material

```
import keras_tuner as kt
def build_model(hp):
    n_hidden = hp.Int("n_hidden", min_value=0, max_value=8, default=2)
    n_neurons = hp.Int("n_neurons", min_value=16, max_value=256)
    learning_rate = hp.Float("learning_rate", min_value=1e-4, max_value=1e-2,
                             sampling="log")
    optimizer = hp.Choice("optimizer", values=["sgd", "adam"])
    if optimizer == "sgd":
        optimizer = keras.optimizers.SGD(learning_rate=learning_rate)
    else:
        optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
    model = keras.Sequential()
    model.add(layers.Flatten())
    for _ in range(n_hidden):
        model.add(layers.Dense(n_neurons, activation="relu"))
    model.add(layers.Dense(10, activation="softmax"))
    model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
                  metrics=["accuracy"])
    return model
```

Extra Material

```
random_search_tuner = kt.RandomSearch(
    build_model, objective="val_accuracy", max_trials=5, overwrite=True,
    directory="my_fashion_mnist", project_name="my_rnd_search", seed=42)
random_search_tuner.search(X_train, y_train, epochs=10,
    validation_data=(X_valid, y_valid))

top3_models = random_search_tuner.get_best_models(num_models=3)
best_model = top3_models[0]

>>> top3_params = random_search_tuner.get_best_hyperparameters(num_trials=3)
>>> top3_params[0].values # best hyperparameter values
{'n_hidden': 5,
 'n_neurons': 70,
 'learning_rate': 0.00041268008323824807,
 'optimizer': 'adam'}
```

9. Fine-Tuning Neural Network Hyperparameters

- **Number of Hidden Layers**
  - One hidden layer can theoretically model even the most complex functions, provided it has enough neurons.
  - But for complex problems, deep networks have a much higher parameter efficiency than shallow ones.
- **Number of Neurons per Hidden Layer**
  - **Pyramid** across layers or **same** size
  - **Stretch pants**: pick a model with more layers and neurons than you need, then use early stopping and other regularization techniques to prevent it from overfitting.
- Better to increase the number of layers instead of the number of neurons per layer.

9. Fine-Tuning Neural Network Hyperparameters

- **Learning Rate**: The optimal LR is about half of the maximum LR.
- **Optimizer**: There are other than the Mini-batch Gradient Descent optimizer.
- **Batch Size**
  - Larger gives better speed up with hardware accelerators.
  - Smaller makes the models more general.
- **Activation Functions**
- **Number of iterations**: Use early stopping.

10. Tutorials

- https://keras.io/
- https://www.tensorflow.org/guide/keras
- Keras Tutorial: Deep Learning in Python from DataCamp, https://www.datacamp.com/community/tutorials/deep-learning-python
- Keras Tutorial: The Ultimate Beginner's Guide to Deep Learning in Python, from EliteDataScience, https://elitedatascience.com/keras-tutorial-deep-learning-in-python

11. Exercise

From Chapter 10, solve exercise:

- 10. Train a deep MLP on the **MNIST** dataset (you can load it using `keras.datasets.mnist.load_data()`. See if you can get over **98%** precision. Try searching for the optimal learning rate by using the approach presented in this chapter (i.e., by growing the learning rate exponentially, plotting the error, and finding the point where the error shoots up). Try adding all the bells and whistles—save checkpoints, use **early stopping**, and plot learning curves using **TensorBoard**.

Summary

1. Introduction
2. Keras API Styles
3. TensorFlow Keras
4. Image Classifier Using the Sequential Model
5. Regression Using the Sequential Model
6. Using the Functional API
7. Using Callbacks
8. Visualization Using TensorBoard
9. Fine-Tuning Neural Network Hyperparameters
10. Tutorials
11. Exercise