Chapter 10 – Introduction to Artificial Neural Networks with Keras

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

[Run in Google Colab (https://colab.research.google.com/github/ageron/handson-ml2/blob/master/10_neural_nets_with_keras.ipynb)](https://colab.research.google.com/github/ageron/handson-ml2/blob/master/10_neural_nets_with_keras.ipynb)

Setup

First, let's import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures. We also check that Python 3.5 or later is installed (although Python 2.x may work, it is deprecated so we strongly recommend you use Python 3 instead), as well as Scikit-Learn ≥0.20 and TensorFlow ≥2.0.

In [1]:
```
# 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

# 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)
```

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

Perceptrons

Note: we set `max_iter` and `tol` explicitly to avoid warnings about the fact that their default value will change in future versions of Scikit-Learn.

In [2]:
```python
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]])
```

```
/home/madhavan/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To s
ilence this warning, use `int` by itself. Doing this will not modify any behav
ior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64`
or `np.int32` to specify the precision. If you wish to review your current us
e, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/dev
docs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release
/1.20.0-notes.html#deprecations)
  import sys
```

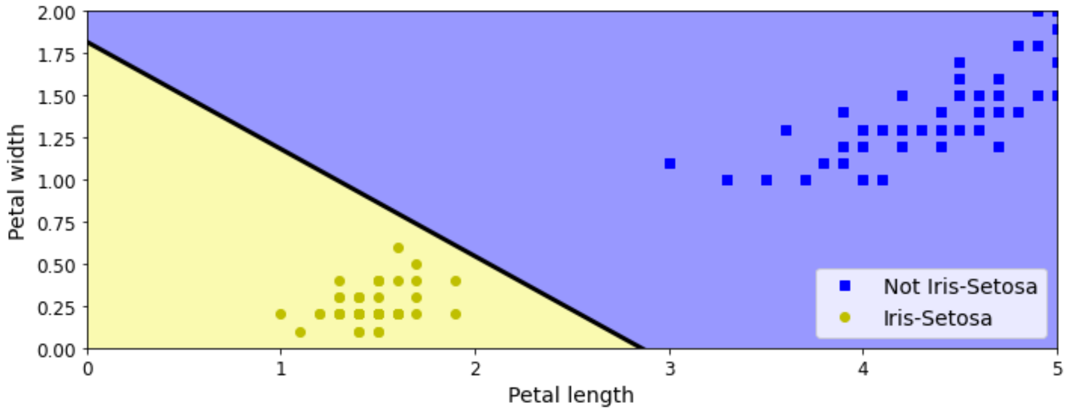
In [3]:
```python
y_pred
```

Out[3]:
```
array([1])
```

In [4]:
```python
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-", linewidt
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
```

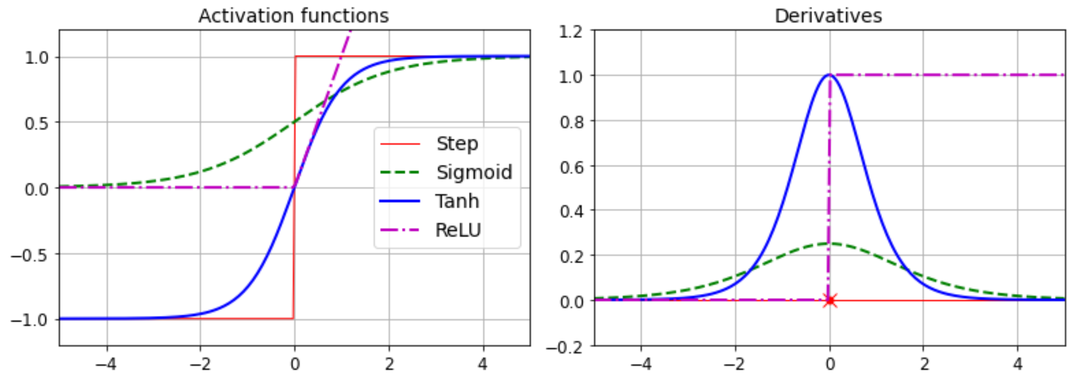


Activation functions

In [5]:
```
def sigmoid(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)
```

In [6]:
```
z = np.linspace(-5, 5, 200)

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

plt.subplot(121)
plt.plot(z, np.sign(z), "r-", linewidth=1, label="Step")
plt.plot(z, sigmoid(z), "g--", linewidth=2, label="Sigmoid")
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=1, label="Step")
plt.plot(0, 0, "ro", markersize=5)
plt.plot(0, 0, "rx", markersize=10)
plt.plot(z, derivative(sigmoid, z), "g--", linewidth=2, label="Sigmoid")
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])

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

```
Saving figure activation_functions_plot
```



Building an Image Classifier

First let's import TensorFlow and Keras.

In [9]:
```
import tensorflow as tf
from tensorflow import keras
```

In [10]:
```
tf.__version__
```

Out[10]:
```
'2.4.1'
```

In [11]:
```
keras.__version__
```

Out[11]:
```
'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 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:

In [12]:
```
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

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

In [13]:
```
X_train_full.shape
```

Out[13]:
```
(60000, 28, 28)
```

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

In [14]:
```
X_train_full.dtype
```

Out[14]:
```
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.

In [15]:
```
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:

In [16]:
```
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:

In [17]:
```
y_train
```

Out[17]:
```
array([4, 0, 7, ..., 3, 0, 5], dtype=uint8)
```

Here are the corresponding class names:

In [18]:
```
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:

In [19]:
```
class_names[y_train[0]]
```
Out[19]:
```
'Coat'
```

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

In [20]:
```
X_valid.shape
```
Out[20]:
```
(5000, 28, 28)
```

In [21]:
```
X_test.shape
```
Out[21]:
```
(10000, 28, 28)
```

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

In [22]:
```
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
```

In [23]:
```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

In [24]:
```
keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

In [25]:
```
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")
])
```

In [26]:
```
model.layers
```

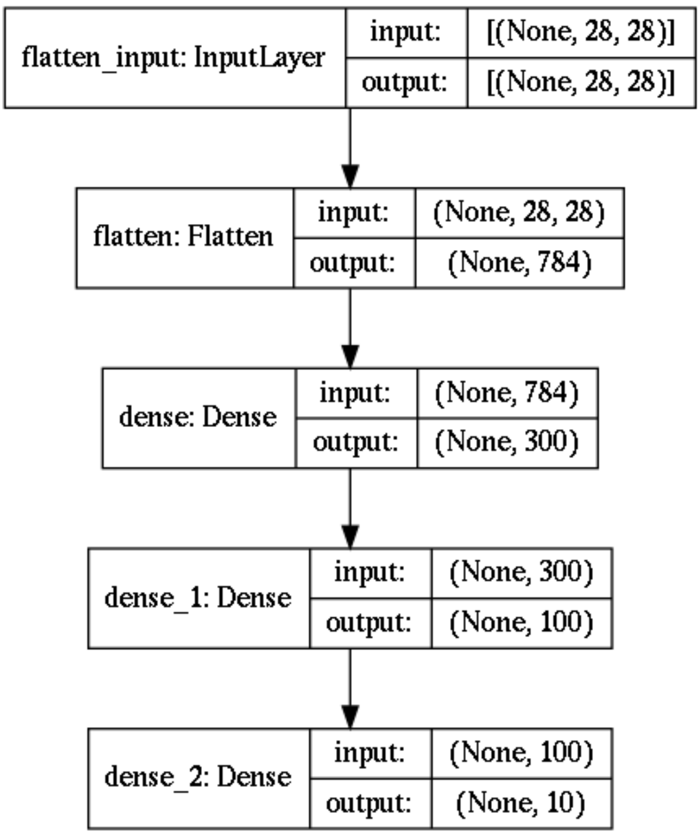
Out[26]:
```
[<tensorflow.python.keras.layers.core.Flatten at 0x7f15546140d0>,
 <tensorflow.python.keras.layers.core.Dense at 0x7f1554614350>,
 <tensorflow.python.keras.layers.core.Dense at 0x7f1554614690>,
 <tensorflow.python.keras.layers.core.Dense at 0x7f155506a310>]
```

In [27]:
```
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
_________________________________________________________________
```

In [28]:
```
keras.utils.plot_model(model, "my_mnist_model.png", show_shapes=True)
```

Out[28]:



In [29]:
```
hidden1 = model.layers[1]
hidden1.name
```
Out[29]:
```
'dense'
```

In [30]:
```
model.get_layer(hidden1.name) is hidden1
```
Out[30]:
```
True
```

In [31]:
```
weights, biases = hidden1.get_weights()
```

In [32]:
```
weights
```
Out[32]:
```
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)
```

In [33]:
```
weights.shape
```
Out[33]:
```
(784, 300)
```

In [34]:
```
biases
```
Out[34]:
```
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)
```

In [35]:
```
biases.shape
```
Out[35]:
```
(300,)
```

In [36]:
```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

This is equivalent to:

```
model.compile(loss=keras.losses.sparse_categorical_crossentropy,
```

In [37]:
```
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```

```
Epoch 1/30
1719/1719 [==============================] - 17s 10ms/step - loss: 1.0188 - ac
curacy: 0.6805 - val_loss: 0.5218 - val_accuracy: 0.8210
Epoch 2/30
1719/1719 [==============================] - 11s 6ms/step - loss: 0.5028 - acc
uracy: 0.8259 - val_loss: 0.4351 - val_accuracy: 0.8530
Epoch 3/30
1719/1719 [==============================] - 15s 9ms/step - loss: 0.4483 - acc
uracy: 0.8423 - val_loss: 0.5354 - val_accuracy: 0.7994
Epoch 4/30
1719/1719 [==============================] - 17s 10ms/step - loss: 0.4207 - ac
curacy: 0.8525 - val_loss: 0.3917 - val_accuracy: 0.8656
Epoch 5/30
1719/1719 [==============================] - 16s 9ms/step - loss: 0.4059 - acc
uracy: 0.8587 - val_loss: 0.3748 - val_accuracy: 0.8688
Epoch 6/30
1719/1719 [==============================] - 12s 7ms/step - loss: 0.3753 - acc
uracy: 0.8678 - val_loss: 0.3711 - val_accuracy: 0.8724
Epoch 7/30
1719/1719 [==============================] - 14s 8ms/step - loss: 0.3651 - acc
uracy: 0.8710 - val_loss: 0.3629 - val_accuracy: 0.8732
Epoch 8/30
1719/1719 [==============================] - 14s 8ms/step - loss: 0.3477 - acc
uracy: 0.8753 - val_loss: 0.3837 - val_accuracy: 0.8626
Epoch 9/30
1719/1719 [==============================] - 19s 11ms/step - loss: 0.3480 - ac
curacy: 0.8760 - val_loss: 0.3596 - val_accuracy: 0.8698
Epoch 10/30
```

In [38]:
```
history.params
```

Out[38]:
```
{'verbose': 1, 'epochs': 30, 'steps': 1719}
```

In [39]:
```
print(history.epoch)
```

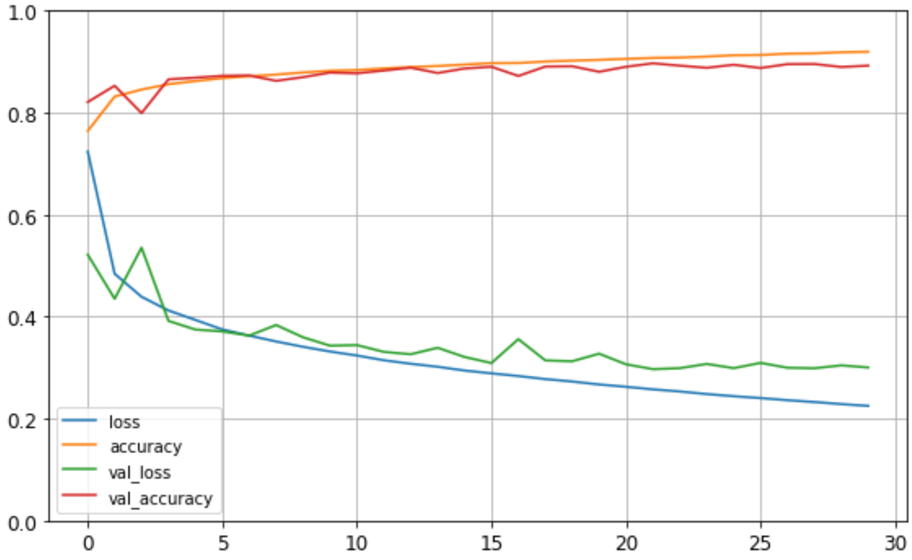
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29]
```

In [40]:
```
history.history.keys()
```

Out[40]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [41]:
```
import pandas as pd

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()
```

```
Saving figure keras_learning_curves_plot
```



In [42]:
```
model.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.3348 - accura
cy: 0.8831
```

Out[42]:
```
[0.33482903242111206, 0.8830999732017517]
```

In [43]:
```
X_new = X_test[:3]
y_proba = model.predict(X_new)
y_proba.round(2)
```

Out[43]:
```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.02, 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)
```

In [44]:
```
y_pred = model.predict_classes(X_new)
y_pred
```

```
/home/madhavan/miniconda3/lib/python3.7/site-packages/tensorflow/python/keras/
engine/sequential.py:450: UserWarning: `model.predict_classes()` is deprecated
and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.pr
edict(x), axis=-1)`,   if your model does multi-class classification   (e.g. i
f it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astyp
e("int32")`,   if your model does binary classification   (e.g. if it uses a `
sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

Out[44]:
```
array([9, 2, 1])
```

In [45]:
```
np.array(class_names)[y_pred]
```

Out[45]:
```
array(['Ankle boot', 'Pullover', 'Trouser'], dtype='<U11')
```

In [46]:
```
y_new = y_test[:3]
y_new
```

Out[46]:
```
array([9, 2, 1], dtype=uint8)
```

In [47]:
```
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
```