3장 *Getting started with neural networks*

"기회와 준비가 만났을 때 ... "

Outline

- Core components of neural networks
- An introduction to Keras
- Setting up a deep-learning workstation
- Using neural networks to solve basic classification and regression problems

Outline

- Classifying movie reviews as positive or negative (binary classification)
- Classifying news wires by topic (multiclass classification)
- Estimating the price of a house, given real-estate data (regression)

1. Anatomy of a neural network

- Relationship between the network, layers, loss function, and optimizer

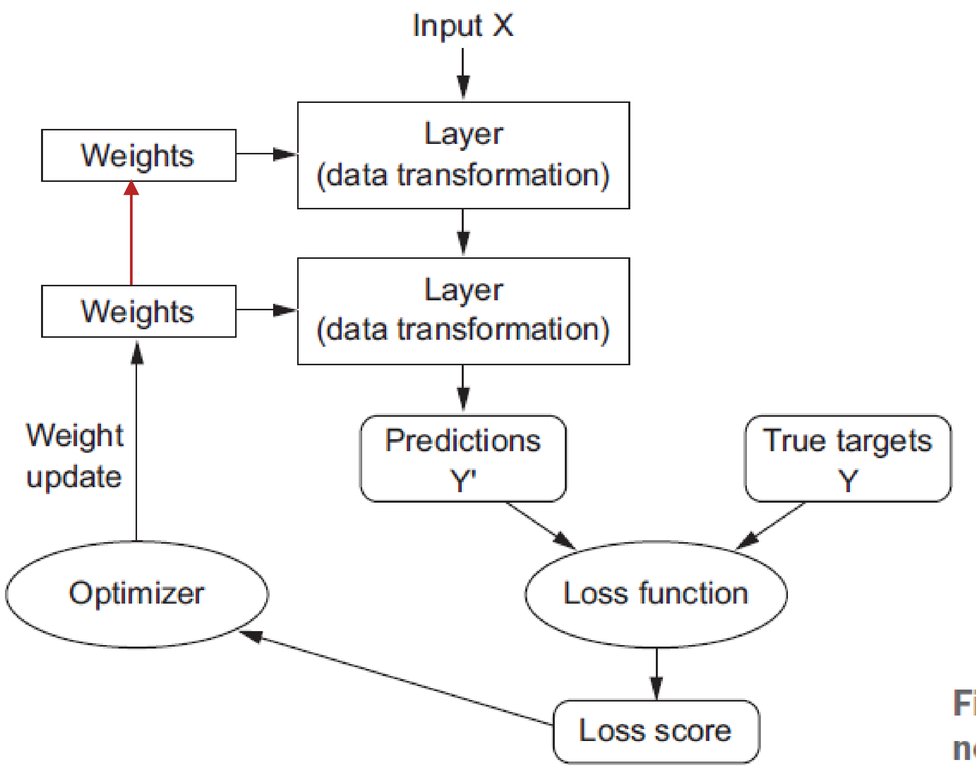


Figure 3.1 Relationship between the network, layers, loss function, and optimizer

1. Anatomy of a neural network

3.1.1 Layers: the building blocks of deep learning

- a data-processing module
- input and outputs tensors
- *weights* tensors contain the network's *knowledge* - *학습결과*
- densely connected layers, fully connected, or dense layers (the Dense class in Keras)

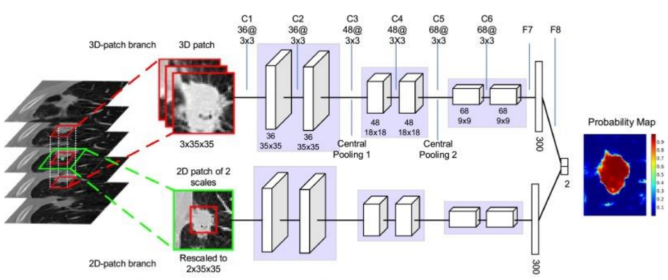
3.1.1 Layers: the building blocks of deep learning

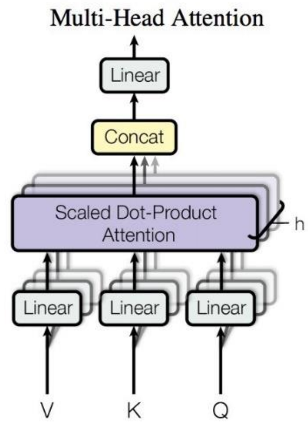
A dense layer with 10 output units

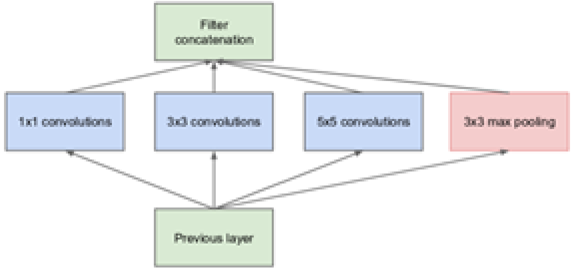
```
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(32,input_shape=(784,)))
model.add(layers.Dense(10))
```

3.1.2 Models: networks of layers

- A deep-learning model is a directed, acyclic graph of layers.
- Search a good set of values for the weight tensors
- Some common topologies include the following:
  - Two-branch networks
  - Multihead networks
  - Inception blocks





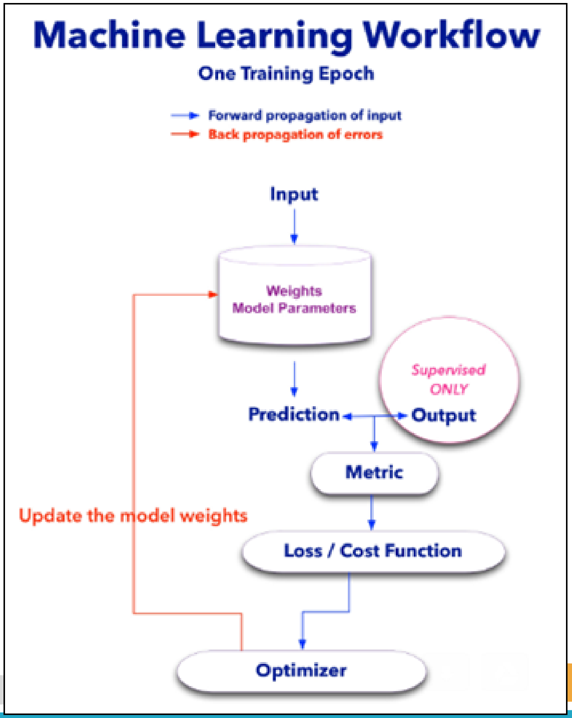


Inception module, naïve version

1. Anatomy of a neural network

3.1.3 Loss functions and optimizers: keys to configuring the learning process

- choose two more things:
  - *Loss function (objective function)*—The quantity that will be minimized during training. It represents a measure of success for the task at hand.
  - *Optimizer*—Determines how the network will be updated based on the loss function. It implements a specific variant of stochastic gradient descent (SGD).



2. Introduction to Keras

- code examples use Keras (https:// keras.io)
- Keras is a deep-learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model.
- Keras was initially developed for researchers, with the aim of enabling fast experimentation.
- Keras has the following key features:
  - same code on CPU or GPU.
  - It has a user-friendly API that makes it easy to quickly prototype deep-learning models.
  - It has built-in support for convolutional networks, recurrent networks
  - It supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, and so on.

2. Introduction to Keras

- Keras is distributed under the permissive MIT license, which means it can be freely used in commercial projects.
- Keras is used at Google, Netflix, Uber, CERN, Yelp, Square, and hundreds of startups working on a wide range of problems.
- Keras is also a popular framework on Kaggle, the machine-learning competition website, where almost every recent deep-learning competition has been won using Keras models.

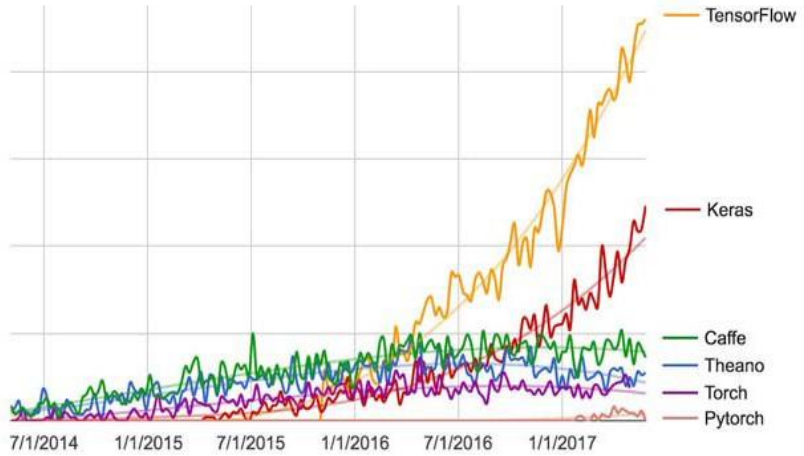


Figure 3.2 Google web search interest for different deep-learning frameworks over time

2. Introduction to Keras

3.2.1 Keras, TensorFlow, Theano, and CNTK

- Keras is a model-level library, providing high-level building blocks for developing deep-learning models.
- the three existing backend implementations - the TensorFlow, the Theano, and the Microsoft Cognitive Toolkit (CNTK) backends. In the future, it's likely that Keras will be extended to work with even more deep-learning execution engines.

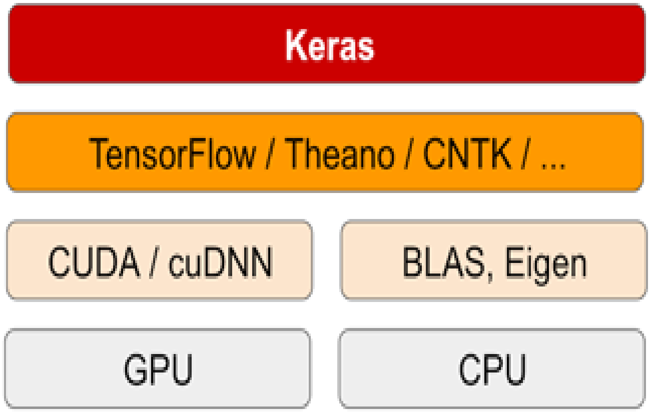


Figure 3.3 The deep-learning software and hardware stack

2. Introduction to Keras

3.2.1 Developing with Keras: a quick overview

- The typical Keras workflow looks just like that example:
  1. Define your training data: input tensors and target tensors.
  2. Define a network of layers (or *model*) that maps your inputs to your targets
  3. Configure the learning process by choosing a loss function, an optimizer, and some metrics to monitor.
  4. Iterate on your training data by calling the `fit()` method of your model.
- There are two ways to define a model: `Sequential` class, the *functional API* (for directed acyclic graphs of layers, which lets you build completely arbitrary architectures).

3.2.1 Developing with Keras: a quick overview

- a two-layer model defined using the `Sequential` class :

```
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(32, activation='relu',
                                              input_shape=(784,)))
model.add(layers.Dense(10,activation='softmax'))
```

2. Introduction to Keras

3.2.1 Developing with Keras: a quick overview

- in the compilation step, where you specify the optimizer and loss function(s)

```
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
    loss='mse', metrics=['accuracy'])
```

- the `fit()` method:

```
model.fit(input_tensor, target_tensor, batch_size=128, epochs=10)
```

3.4 Classifying movie reviews: a binary classification example

- Two-class classification, or binary classification - classify movie reviews as positive or negative, based on the text content of the reviews

3.4.1 The IMDB dataset

- IMDB dataset: a set of 50,000 highly polarized reviews from the Internet Movie Database.
- They're split into 25,000 reviews for training and 25,000 reviews for testing, each set consisting of 50% negative and 50% positive reviews.
- IMDB dataset: the reviews (sequences of words) into sequences of integers

2. Introduction to Keras

3.4.1 The IMDB dataset

- The following code will load the dataset (when you run it the first time, about 80 MB of data will be downloaded to your machine).

Listing 3.1 Loading the IMDB dataset

```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels)
     = imdb.load_data( num_words=10000)
# top 10,000 most frequently occurring words in the training data
>>> train_data[0]
[1, 14, 22, 16, ... 178, 32]    # list of word indices
>>> train_labels[0]
1       # 0 stands for negative and 1 stands for positive
```

2. Introduction to Keras

3.4.1 The IMDB dataset

- no word index will exceed 10,000:

```
>>> max([max(sequence) for sequence in train_data])
9999
```

- back to English words:

```
word_index = imdb.get_word_index()
reverse word index = dict(
                                                  items()])
```

```
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
print(sorted(reverse_word_index.items()))
```

```
[(1, 'the'), (2, 'and'), (3, 'a'), (4, 'of'), (5, 'to'), (6, 'is'), (7, 'br'), (8
'in'), (9, 'it'), (10, 'i'), (11, 'this'), (12, 'that'), (13, 'was'), (14, 'as'),
(15, 'for'), (16, 'with'), (17, 'movie'), (18, 'but'), (19, 'film'), (20, 'on'),
```

```
decoded_review = ' '.join(
    [reverse_word_index.get(i - 3, '?') for i in train_data[0]])
```

```
decoded_review = ' '.join([reverse_word_index.get(i-3, '?') for i in train_data[0]])
print(decoded_review)
```

```
? this film was just brilliant casting location scenery story direction everyone's r
eally suited the part they played and you could just imagine being there robert ? is
an amazing actor and now the same being director ? father came from the same scottis
h island as myself so i loved the fact there was a real connection with this film th
```

3.4.2 Preparing the data

- turn your lists into tensors - vectorize the data

```
[1, 14, 22, 16, ... 178, 32] review words →
[0., 1., 1., ..., 0., 0., 0.] 신경망의 입력을 위한 일정한 10,000개 원소로 된 벡터
```

Listing 3.2 Encoding the integer sequences into a binary matrix

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

>>> x_train[0]
array([ 0., 1., 1., ..., 0., 0., 0.])
```

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Each word gets a 1x9 vector representation

- vectorize labels:

```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

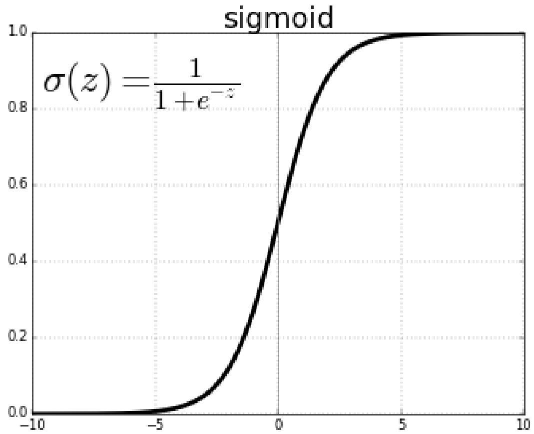
2. Introduction to Keras

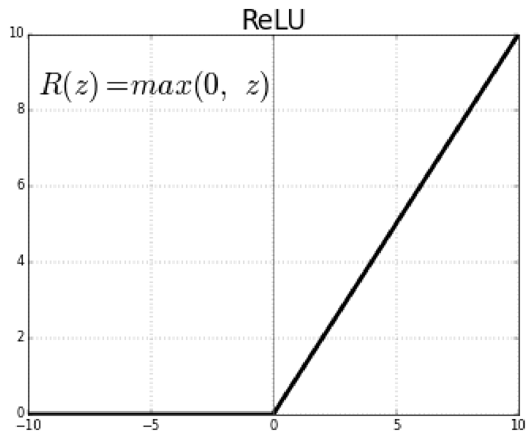
3.4.3 Building your network

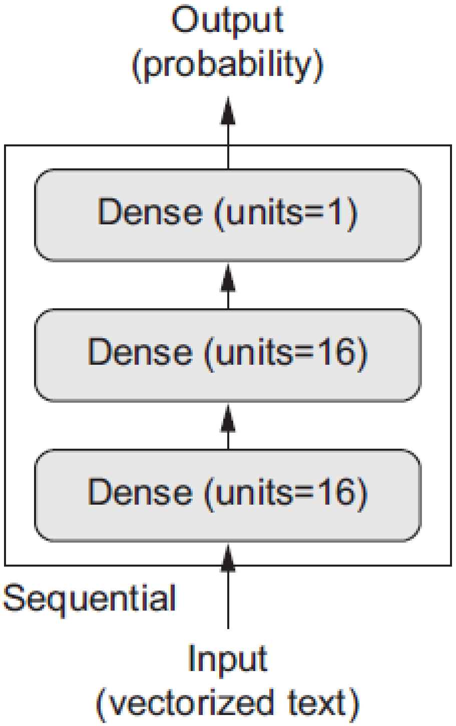
- **Listing 3.3 The model definition**

```
from keras import models from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(1000
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```



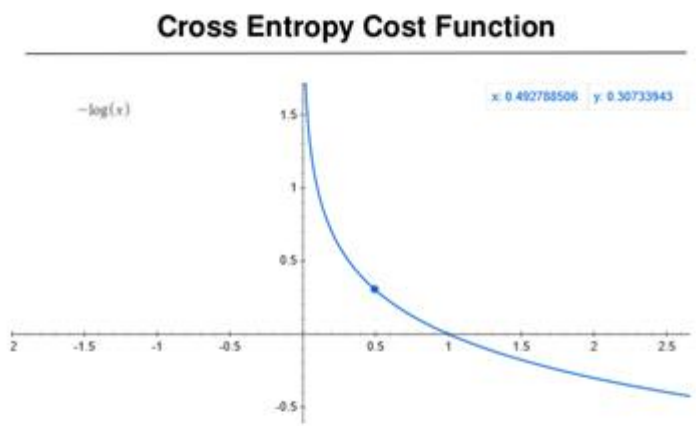




3.4.3 Building your network

Listing 3.4 Compiling the model

```
model.compile(optimizer='rmsprop',
    loss='binary_crossentropy',
    metrics=['accuracy'])
```



$$D(\overline{Y}_i, Y_i) = -\sum Y_i \log \overline{Y_i}$$

$$\begin{bmatrix} \overline{Y_A} \\ \overline{Y_B} \\ \overline{Y_C} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} Y_A \\ Y_B \\ Y_C \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$-\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \cdot \log \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = -\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ \infty \\ \infty \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = 0$$

$$CCE = -\frac{1}{N} \sum_{i=0}^{N} \sum_{j=0}^{J} y_j \cdot log(\hat{y}_j) + (1 - y_j) \cdot log(1 - \hat{y}_j)$$

$$BCE = -\frac{1}{N} \sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

$$CE = -\sum_{i=1}^{n} y_i \log \hat{y}_i$$

3.4.3 Building your network

Listing 3.5 Configuring the optimizer

```
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
     loss='binary_crossentropy', metrics=['accuracy'])
```

Listing 3.6 Using custom losses and metrics

```
from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
        loss=losses.binary_crossentropy,
        metrics=[metrics.binary_accuracy])
```

3.4.3 Building your network

Listing 3.5 Configuring the optimizer

```
from keras import optimizers
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
     loss='binary_crossentropy', metrics=['accuracy'])
```

Listing 3.6 Using custom losses and metrics

```
from keras import losses
from keras import metrics
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
        loss=losses.binary_crossentropy,
        metrics=[metrics.binary_accuracy])
```

3.4.5 Validating your approach

- create a validation set by setting apart 10,000 samples from the original training data.

Listing 3.7 Setting aside a validation set

```
x_val = x_train[:10000]              # 검증 데이터
partial_x_train = x_train[10000:] # 훈련 데이터
y_val = y_train[:10000]              # 검증 lable
partial_y_train = y_train[10000:] # 훈련 lable
```

Listing 3.9 Training your model

```
model.compile(optimizer='rmsprop',
     loss='binary_crossentropy', metrics=['acc'])

history = model.fit(partial_x_train,
        partial_y_train, epochs=20, batch_size=512,
        validation_data=(x_val, y_val))
```

```
results
```

```
[0.9122337437653542, 0.85004]
```

3.4.4 Validating your approach

- create a validation set by setting apart 10,000 samples from the original training data.

Listing 3.7 Setting aside a validation set

```
x_val = x_train[:10000]             # 검증 데이터
partial_x_train = x_train[10000:] # 훈련 데이터
y_val = y_train[:10000]             # 검증 lable
partial_y_train = y_train[10000:] # 훈련 lable
```

Listing 3.9 Training your model

```
model.compile(optimizer='rmsprop',
      loss='binary_crossentropy', metrics=['acc'])

history = model.fit(partial_x_train,
         partial_y_train, epochs=20, batch_size=512,
         validation_data=(x_val, y_val))
```

```
results
```

```
[0.9122337437653542, 0.85004]
```

```
>>> history_dict = history.history # model.fit 훈련정보를 dictionary에 반환
>>> history_dict.keys()
[u'acc', u'loss', u'val_acc', u'val_loss']
```

3.4.5 Validating your approach

- create a validation set by setting apart 10,000 samples from the original training data.

Listing 3.7 Setting aside a validation set

```
x_val = x_train[:10000]               # 검증 데이터
partial_x_train = x_train[10000:] # 훈련 데이터
y_val = y_train[:10000]               # 검증 lable
partial_y_train = y_train[10000:] # 훈련 lable
```

Listing 3.8 Training your model

```
model.compile(optimizer='rmsprop',
     loss='binary_crossentropy', metrics=['acc'])

history = model.fit(partial_x_train,
          partial_y_train, epochs=20, batch_size=512,
          validation_data=(x_val, y_val))
```

```
results
```

```
[0.9122337437653542, 0.85004]
```

```
>>> history_dict = history.history # model.fit 훈련정보를 dictionary에 반환
>>> history_dict.keys()
[u'acc', u'loss', u'val_acc', u'val_loss']
```

3.4.4 Validating your approach

- Matplotlib to plot the training and validation loss side by side (see figure 3.7), and the training and validation accuracy (see figure 3.8)

Listing 3.9 Plotting the training and validation loss

```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

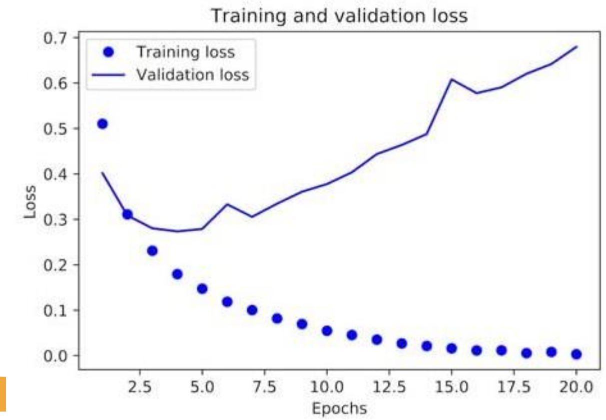


Figure 3.7 Training and validation loss

3.4.4 Validating your approach

- Matplotlib to plot the training and validation loss side by side (see figure 3.7), and the training and validation accuracy (see figure 3.8)

Listing 3.10 Plotting the training and validation accuracy

```
plt.clf()
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend()
plt.show()
```

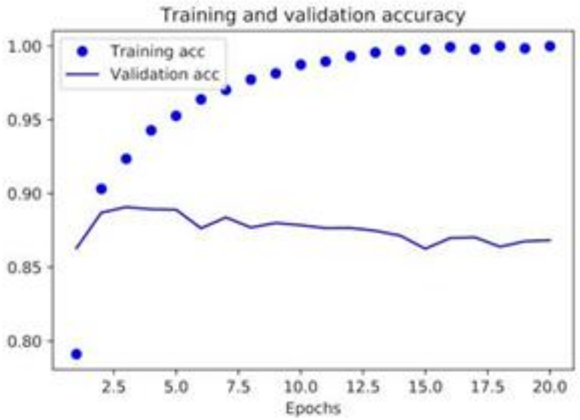


Figure 3.8 Training and validation accuracy

2. Introduction to Keras

3.4.4 Validating your approach

- *overfitting*: specific to the training data and don't generalize to data outside of the training set

Listing 3.11 Retraining a model from scratch

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
     loss='binary_crossentropy', metrics=['accuracy'])

model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

- The final results are as follows:

```
>>> results
[0.2929924130630493, 0.88327999999999995]
```

2. Introduction to Keras

3.4.7 Wrapping up

- preprocessing on your raw data in order to be able to feed it—as tensors—into a neural network. Sequences of words can be encoded as binary vectors.
- Stacks of `Dense` layers with `relu` activations can solve a wide range of problems (including sentiment classification).
- In a binary classification problem (two output classes) - end with one unit `Dense` layer and a `sigmoid` activation: the output of your network should be a scalar between 0 and 1, encoding a probability.
- a binary classification problem - use `binary_crossentropy` loss function
- The `rmsprop` optimizer is generally a good enough choice
- overfitting - worse results on data they've never seen before. Be sure to always monitor performance on data that is outside of the training set.