3장 *Getting started with neural networks*

"기회와 준비가 만났을 때 ... "

5. Classifying newswires: a multiclass classification example

- more than two classes?
- classify Reuters newswires into 46 mutually exclusive topics - *multi-class classification*

3.5.1 The Reuters dataset

- a set of short newswires and their topics, published by **Reuters** in 1986 for text classification
- 46 different topics; each topic has at least 10 examples in the training set
- Like IMDB and MNIST, the **Reuters** dataset comes packaged as part of Keras.

5. Classifying newswires: a multiclass classification example

Listing 3.12 Loading the Reuters

```
from keras.datasets import reuters
(train_data, train_labels), (test_data, test_labels) =
    reuters.load_data(num_words=10000)
```

- num_words=10000 restricts the data to the 10,000 most frequently occurring words found in the data
- 8,982 training examples and 2,246 test examples:

```
>>> len(train_data) 8982
>>> len(test_data) 2246
```

- each example is a list of integers (word indices):

```
>>> train_data[10]
[1, 245, 273, 207, 156, 53, 74, 160, 26, 14, 46, 296, 26, 39, 74, 2979,3554,
14, 46, 4689, 4329, 86, 61, 3499, 4795, 14, 61, 451, 4329, 17, 12]
```

- label : 0 and 45, a topic index

```
>>> train_labels[10]
3
```

3.5.2 Preparing the data

- vectorize the data - 46 different topics; each topic has at least 10 examples in the training set

Listing 3.14 Encoding the data

```
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)
```

3.5.2 Preparing the data

- One-hot encoding (*categorical encoding*) - an all-zero vector with a 1 in the place of the label index. Here's an example:

```
def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results
one_hot_train_labels = to_one_hot(train_labels)
one_hot_test_labels = to_one_hot(test_labels)
```

- built-in way to do this in Keras, in the MNIST example:

```
from keras.utils.np_utils import to_categorical

one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

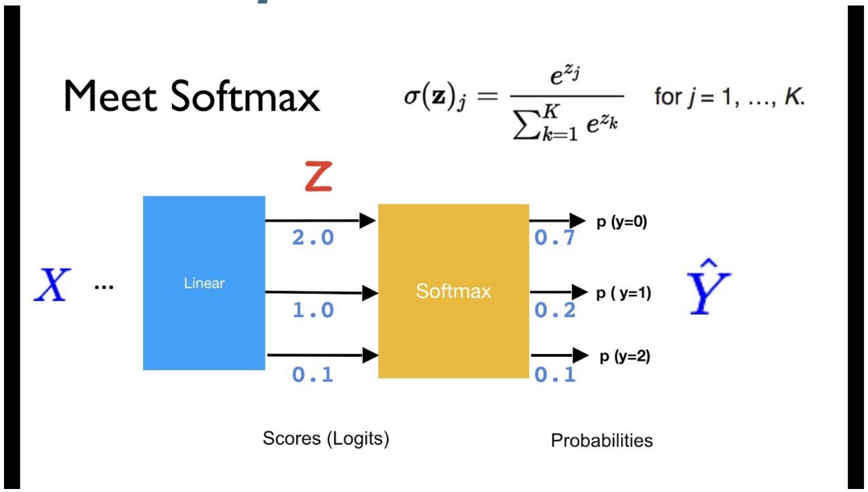
5. Classifying newswires: a multiclass classification example

3.5.3 Building your network

- the number of output classes - 46
- use larger layers with 64 units

Listing 3.15 Model definition

```
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(64, activation='relu',
    input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
```



- There are two other things you should note about this architecture:
  - the network will output a 46-dimensional vector
  - The last layer uses a `softmax` activation - a *probability distribution* over the 46 different output classes — `output[i]` is the probability that the sample belongs to class `i`. The 46 scores will sum to 1.

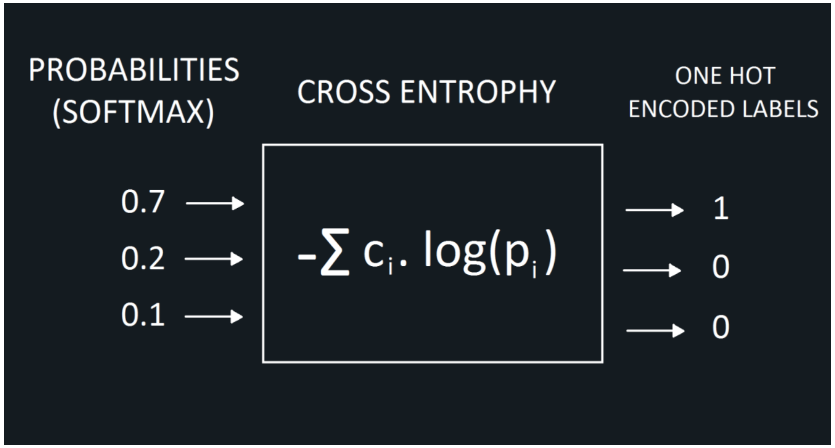
5. Classifying newswires: a multiclass classification example

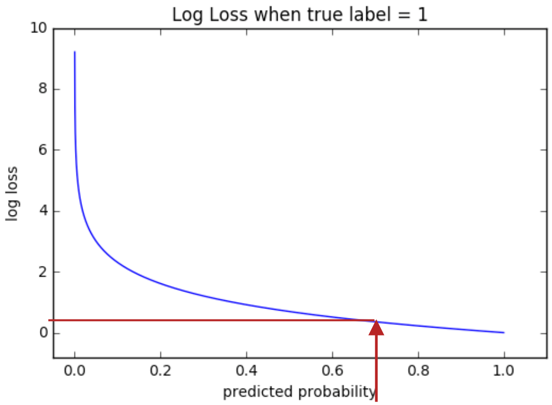
3.5.3 Building your network

- loss function - categorical_crossentropy, distance between the probability distribution output by the network and the true distribution of the labels, minimizing the distance

Listing 3.16 Compiling the model

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





3.5.4 Validating your approach

- Let's set apart 1,000 samples in the training data to use as a validation set.

Listing 3.17 Setting aside a validation set

```
x_val = x_train[:1000]
partial_x_train = x_train[1000:] #7982

y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
```

- Now, let's train the network for 20 epochs.

Listing 3.18 Training the model

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

5. Classifying newswires: a multiclass classification example

3.5.4 Validating your approach

- let's display its loss and accuracy curves (see figures 3.9 and 3.10)

Listing 3.19 Plotting the training and validation loss

```
import matplotlib.pyplot as plt
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

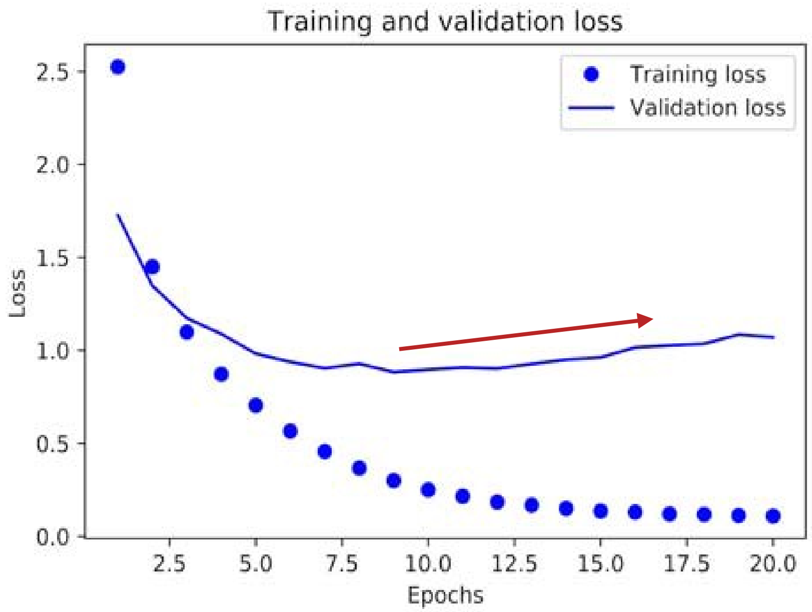


Figure 3.9 Training and validation loss

5. Classifying newswires: a multiclass classification example

3.5.4 Validating your approach

- let's display its loss and accuracy curves (see figures 3.9 and 3.10)

Listing 3.20 Plotting the training and validation accuracy

```
plt.clf()
acc = history.history['acc']
val_acc = history.history['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('Loss')
plt.legend()
plt.show()
```

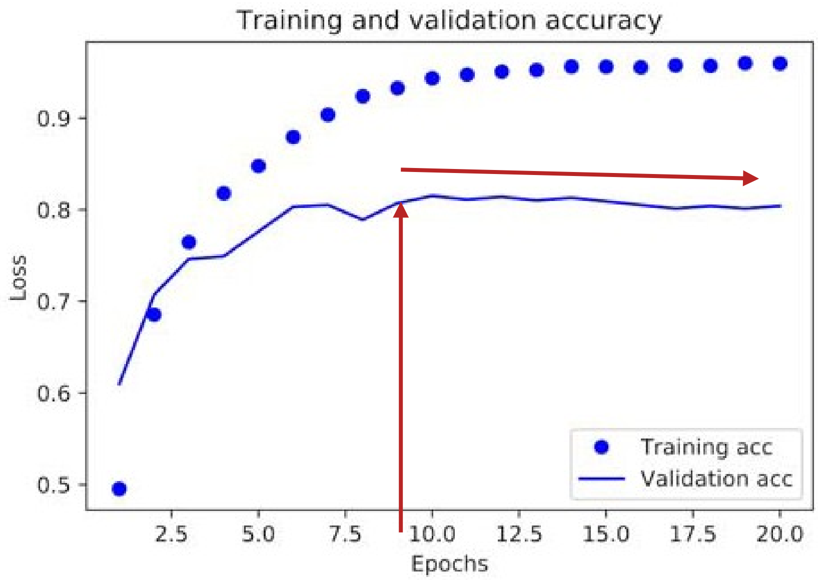


Figure 3.10 Training and validation accuracy

5. Classifying newswires: a multiclass classification example

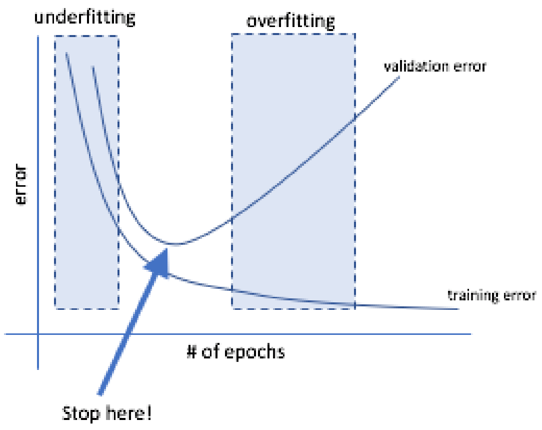
3.5.4 Validating your approach

- The network begins to overfit after **nine** epochs. Let's train a new network from scratch for nine epochs and then evaluate it on the test set.

Listing 3.21 Retraining a model from scratch

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
     loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(partial_x_train,
        partial_y_train, epochs=9, batch_size=512,
        validation_data=(x_val, y_val))
results = model.evaluate(x_test, one_hot_test_labels)
```



- Here are the final results:

```
>>> results
[0.9565213431445807, 0.7969723953695458 9]
```

5. Classifying newswires: a multiclass classification example

3.5.5 Generating predictions on new data

- You can verify that the `predict` method of the model instance returns a probability distribution over all 46 topics. Let's generate topic predictions for all of the test data.

Listing 3.22 Generating predictions for new data

```
predictions = model.predict(x_test)
```

- Each entry in `predictions` is a vector of length 46:

```
>>> predictions[0].shape
(46,)
```

- The coefficients in this vector sum to 1:

```
>>> np.sum(predictions[0])
1.0
```

The largest entry is the predicted class:

```
>>> np.argmax(predictions[0]) # 1.0이 있는 위치의 index 값
4
```

```
[0., 0., 0., 0., 1., 0., … , 0.]
```

5. Classifying newswires: a multiclass classification example

3.5.6 A different way to handle the labels and the loss

- We mentioned earlier that another way to encode the labels would be to cast them as an integer tensor, like this:

```
y_train = np.array(train_labels) #
y_test = np.array(test_labels) #
```

- loss function – `categorical_crossentropy` → `sparse_categorical_crossentropy`:

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

If your targets are **one-hot encoded**, use

categorical_crossentropy

index	normal	neptune	smurf
0	1	0	0
1	0	1	0
2	0	0	1
3	1	0	0
4	1	0	0
5	1	0	0

But if your targets are **integers**, use

sparse_categorical_crossentropy

normal = 1
neptune= 2
smurf= 3

index	y
0	1
1	2
2	3
3	1
4	1
5	1

5. Classifying newswires: a multiclass classification example

3.5.7 The importance of having sufficiently large intermediate layers

- final outputs are 46-dimensional, intermediate layers << 46-dimensional: for example, 4-dimensional.

Listing 3.23 A model with an information bottleneck

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(4, activation='relu')) # too low-dimensional → 32, 128, …
model.add(layers.Dense(46, activation='softmax')) #  hyperplanes of 46 classes
model.compile(optimizer='rmsprop',
     loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(partial_x_train,
   partial_y_train, epochs=20, batch_size=128,
   validation_data=(x_val, y_val))
```

- The network now peaks at ~71% validation accuracy, an 8% absolute drop.

3.5.8 Further experiments

- Try using larger or smaller layers: 32 units, 128 units, and so on.
- You used two hidden layers. Now try using a single hidden layer, or three hidden layers.

3.5.9 Wrapping up

- Here's what you should take away from this example:

- classify *N* classes, `Dense` output layer of size *N*.
- multiclass classification problem - `softmax` activation to output a probability distribution over the *N* output classes.
- categorical crossentropy - minimizes the distance between the probability distributions output
- There are two ways to handle labels in multiclass classification:
  – one-hot encoding using `categorical_crossentropy` as a loss function
  – the labels as integers using the `sparse_categorical_crossentropy` loss function
- multiclass classification problem - avoid creating information bottlenecks