CH4. Getting Started with Neural Networks

Il-Youp Kwak

Chung-Ang University

Getting Started with Neural Networks

Common ML examples

1. Classifying movie reviews: a binary classification example
2. Classifying newswires: a multiclass classification example
3. Predicting house prices: a regression example

Classifying movie reviews: a binary classification example

The IMDB dataset:

- **50,000 highly polarized reviews**
- **25,000 reviews for training and 25,000 reviews for testing**
- **Each set consisting of 50% negative and 50% positive reviews**

데이터 균형 맞아야 함!

- **Read the data**

Listing 4.1 Loading the IMDB dataset

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

↳ this dataset is included in keras.dataset

we'll use ↳ this number of frequently used words in review.

- **Decode a review**

Listing 4.2 Decoding reviews back to text

```
word_index = imdb.get_word_index()
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])
decoded_review = " ".join(
    [reverse_word_index.get(i - 3, "?") for i in train_data[0]])
```

word_index is a dictionary mapping words to an integer index.

Reverses it, mapping integer indices to words

[11, 3, 90, 101]

iterate

this would be squence of numbers

eg) [11, 3, 90, 101]

Decodes the review. Note that the indices are offset by 3 because 0, 1, and 2 are reserved indices for "padding," "start of sequence," and "unknown."

This is a companion notebook for the book Deep Learning with Python, Second Edition. For readability, it only contains runnable code blocks and section titles, and omits everything else in the book: text paragraphs, figures, and pseudocode.

If you want to be able to follow what's going on, I recommend reading the notebook side by side with your copy of the book.

This notebook was generated for TensorFlow 2.6.

Getting started with neural networks: Classification and regression

Classifying movie reviews: A binary classification example

The IMDB dataset

Loading the IMDB dataset

```
[ ] from tensorflow.keras.datasets import imdb
    (train_data, train_labels), (test_data, test_labels) = imdb.load_data(
        num_words=10000)
```

```
[ ] train_data[0]
```

- # Preparing the data

Listing 4.3 Encoding the integer sequences via multi-hot encoding

For training, we need fixed size of given set.
[3, 1, 9, 2, 5]
[3, 1, 1, 1]
[1, 3, 1, 3, 5, 6]
we should make these dimension same.

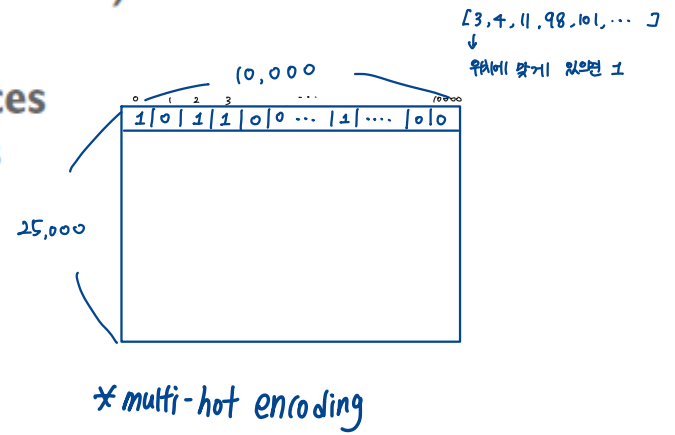
```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        for j in sequence:
            results[i, j] = 1.
    return results
x_train = vectorize_sequences(train_data)
x_test =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

Creates an all-zero matrix of shape (len(sequences), dimension)

Sets specific indices of results[i] to 1s

Vectorized training data

Vectorized test data



Vectorize labels

원래 int type

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

step 1. prepare our data

step 2. building NN model

- # Building your network

Listing 4.4 Model definition

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

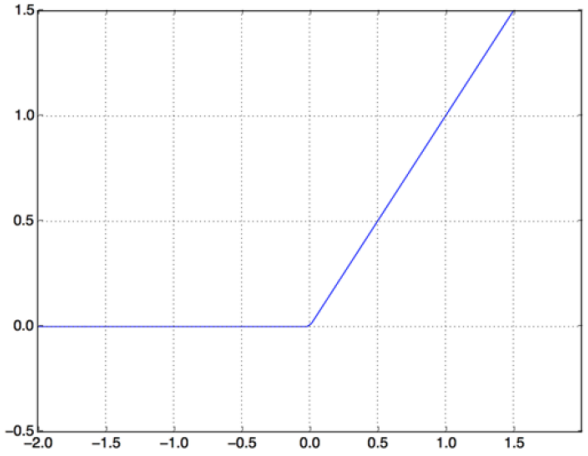
16 output units

max(0,x)

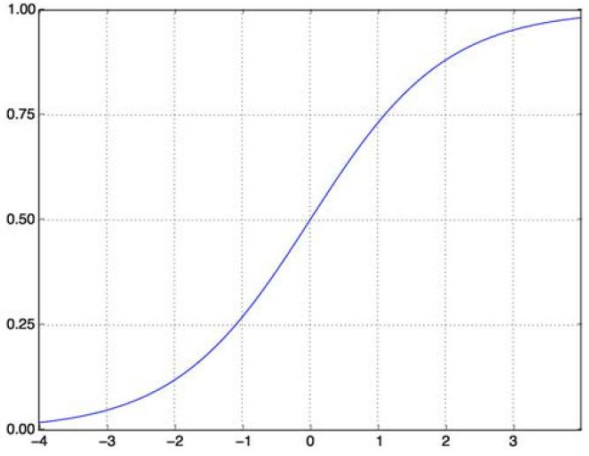
↳ classifying pos/neg

$(-\infty, \infty) \rightarrow [0, 1)$ (neg ↑, pos ↑)

Rectified linear unit (relu)



Sigmoid



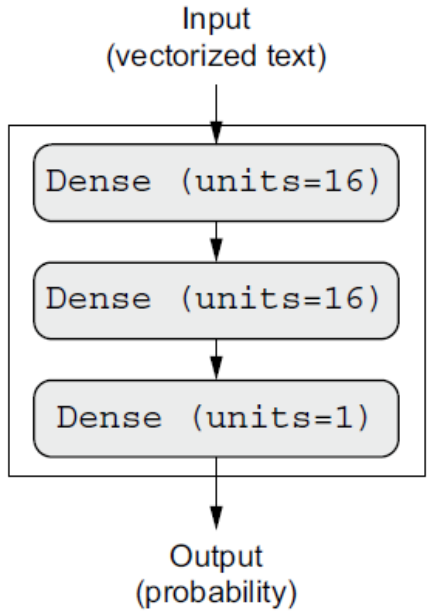
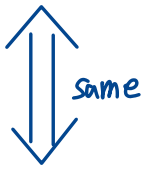


Figure 4.1 The three-layer model

Listing 4.5 Compiling the model

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



- **Configuring optimizer, losses and metrics**

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

- # Validating your approach

Listing 4.6 Setting aside a validation set

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Listing 4.7 Training your model

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

Listing 4.8 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(loss_values) + 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()
```

"bo" is for "blue dot."

"b" is for "solid blue line."

Listing 4.9 Plotting the training and validation accuracy

```
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
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("Accuracy")
plt.legend()
plt.show()
```

Clears the figure

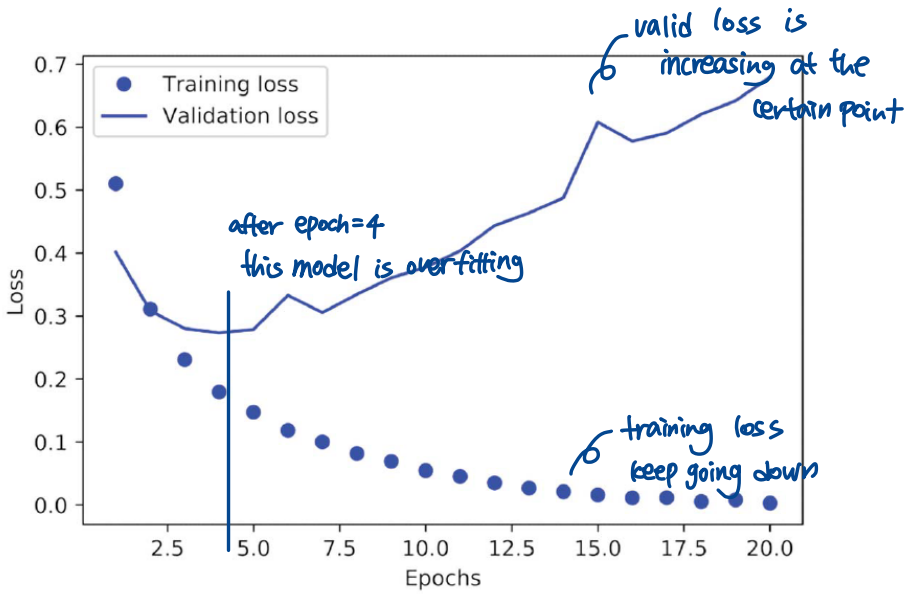


Figure 4.4 Training and validation loss

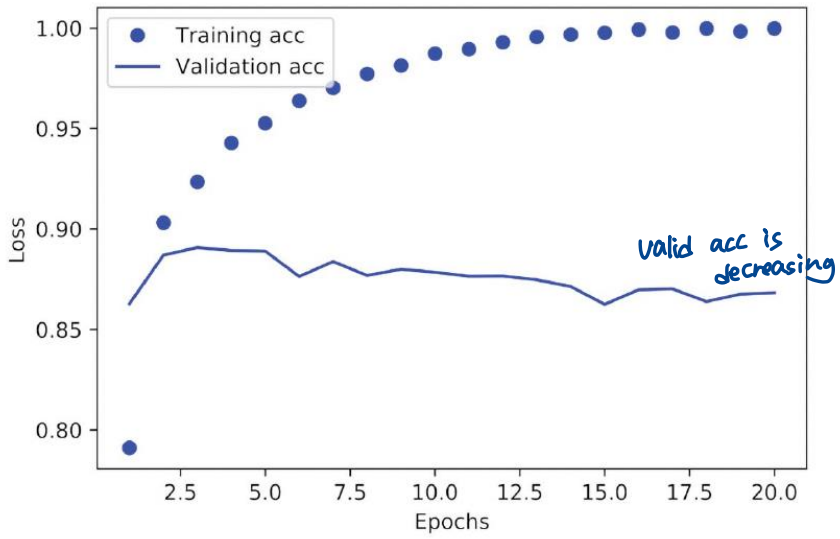


Figure 4.5 Training and validation accuracy

Overfitting: after the fourth epoch, you're overoptimizing on the training data

Listing 4.10 Retraining a model from scratch

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    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, 0.8832799999999995]
```

loss for test set

acc for test

The first number, 0.29, is the test loss, and the second number, 0.88, is the test accuracy.

Using a trained model to generate predictions on new data

```
>>> model.predict(x_test)
array([[ 0.98006207]
       [ 0.99758697]
       [ 0.99975556]
       ...,
       [ 0.82167041]
       [ 0.02885115]
       [ 0.65371346]], dtype=float32)
```

```
    [reverse_word_index.get(i - 3, "?") for i in train_data[0]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1646592/1641221 [==============================] - 0s 0us/step
1654784/1641221 [==============================] - 0s 0us/step
```

```
decoded_review
```

```
'? this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being th
ere robert ? is an amazing actor and now the same being director ? father came from the same scottish island as myself so i loved the fact there was a rea
l connection with this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was releas
ed for ? and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if y
ou cry at a film it must have been good and this definitely was also ? to the two little boy's that played the ? of norman and paul they were just brillia
nt children are often left out of the ? list i think because the stars that play them all grown up are such a big profile for the whole film but these chi
ldren are amazing and should be praised for what they have done don't you thi...'
```

Preparing the data

Encoding the integer sequences via multi-hot encoding

```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        for j in sequence:
            results[i, j] = 1.
```

Exercises :D

- Try using one or three hidden layers, and see how doing so affects validation and test accuracy.
- Try using layers with more hidden units or fewer hidden units
- Try using the mse loss function instead of binary_crossentropy.
- Try using the tanh activation instead of relu.

Classifying newswires: a multiclass classification example

The Reuters dataset:

- **Dataset for text classification (46 different topics)**
- **8,982 training and 2,246 test examples**

Listing 4.11 Loading the Reuters dataset

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

using most frequent 10,000 words

Listing 4.12 Decoding newswires back to text

```
word_index = reuters.get_word_index()
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])
decoded_newswire = " ".join(
    [reverse_word_index.get(i - 3, "?") for i in     train_data[0]])
```

Note that the indices are offset by 3 because 0, 1, and 2 are reserved indices for "padding," "start of sequence," and "unknown."

Listing 4.13 Encoding the input data

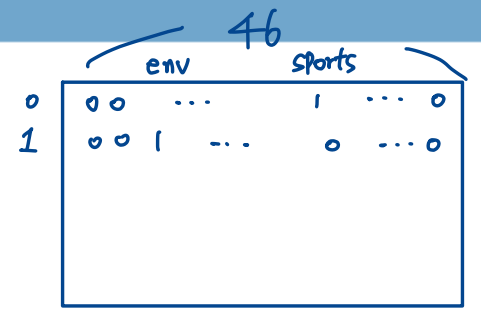
```
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

Vectorized training data

Vectorized test data

Listing 4.14 Encoding the labels

```
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
y_train = to_one_hot(train_labels)
y_test = to_one_hot(test_labels)
```



Vectorized training labels

Vectorized test labels

Can be done using built in function: to_categorical()

사용자정의 함수 안해도 됨

```
from tensorflow.keras.utils import to_categorical
y_train = to_categorical(train_labels)
y_test = to_categorical(test_labels)
```

Building our model

Listing 4.15 Model definition

```
model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(46, activation="softmax")
])
```

46 outcomes

nature expansion of 'sigmoid'

Listing 4.16 Compiling the model

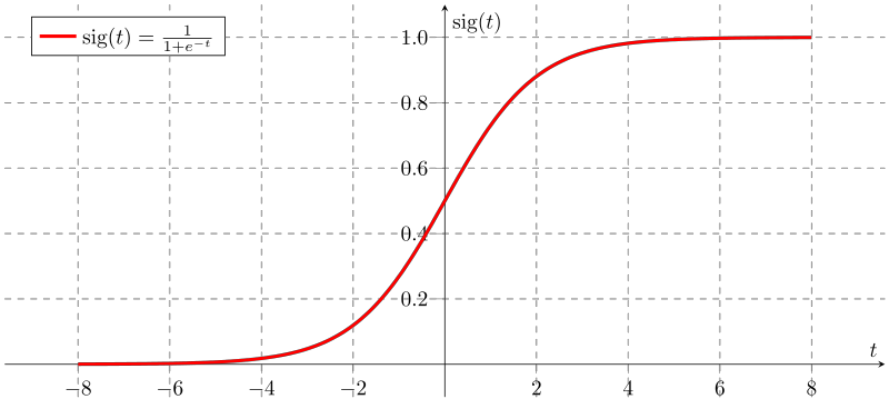
```
model.compile(optimizer="rmsprop",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

For binary classification, we use sigmoid with single dense output layer

```
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'))
```

0 → Negative

1 → Positive



If we use Dense layer with output 2, natural extension should be..

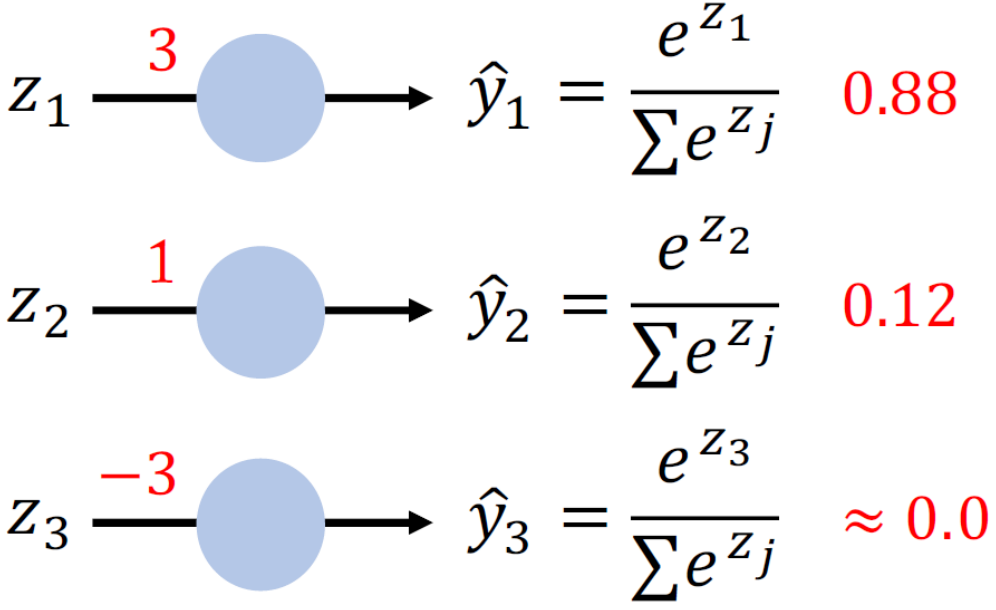
$$\begin{Bmatrix} \frac{e^{-t}}{1+e^{-t}} \\ \frac{1}{1+e^{-t}} \end{Bmatrix} = \begin{Bmatrix} \frac{e^{-z_1}}{e^{-z_1}+e^{-z_2}} \\ \frac{e^{-z_2}}{e^{-z_1}+e^{-z_2}} \end{Bmatrix}$$

$$\frac{e^{z1}}{e^{z1}+e^{z2}} = \frac{1}{1+e^{z2-z1}}$$

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(2, activation='softmax'))
```

Softmax function for multi-class classification

Softmax layer : extension of sigmoid



Probability shape: $0 \leq \hat{y}_i \leq 1$

$$\sum \hat{y}_i = 1$$

Listing 4.17 Setting aside a validation set

```
x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = y_train[:1000]
partial_y_train = y_train[1000:]
```

Listing 4.18 Training the model

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

Listing 4.19 Plotting the training and validation loss

```
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()
```

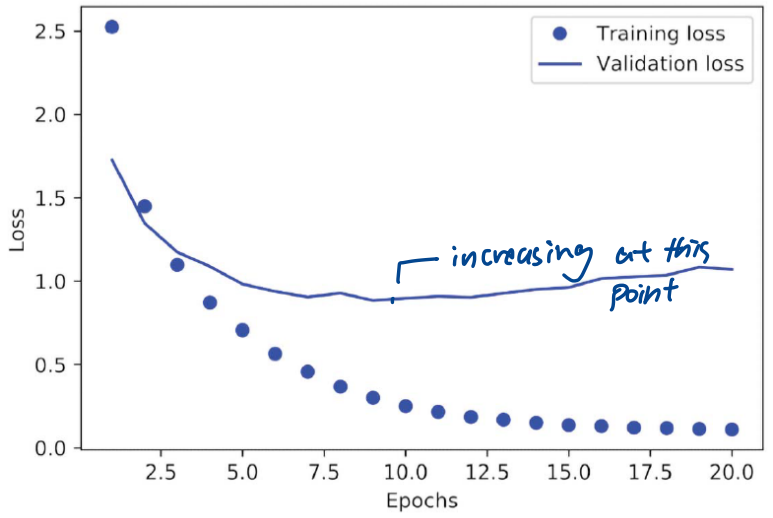


Figure 4.6 Training and validation loss

Listing 4.20 Plotting the training and validation accuracy

```
plt.clf()      <-- Clears the figure
acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training accuracy")
plt.plot(epochs, val_acc, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

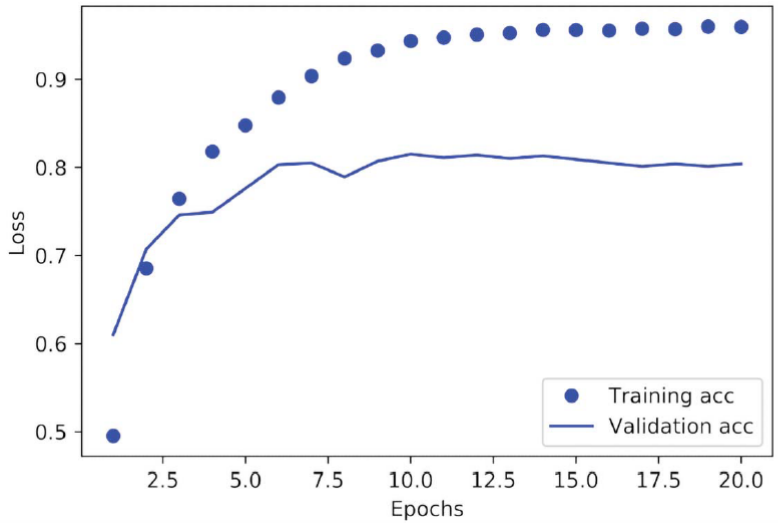


Figure 4.7 Training and validation accuracy

The model begins to overfit after nine epochs.

Listing 4.21 Retraining a model from scratch

```
model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(46, activation="softmax")
])
model.compile(optimizer="rmsprop",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
model.fit(x_train,
          y_train,
          epochs=9,
          batch_size=512)
results = model.evaluate(x_test, y_test)
```

Final result

```
>>> results                                    acc : 79.7 %
[0.9565213431445807, 0.7969723953695458]
```

Result from random guess

```
>>> import copy
>>> test_labels_copy = copy.copy(test_labels)
>>> np.random.shuffle(test_labels_copy)
>>> hits_array = np.array(test_labels) == np.array(test_labels_copy)
>>> hits_array.mean()
0.18655387355298308
```

A random classifier would score around 19% classification accuracy, so the results of our model seem pretty good in that light.

Generating predictions on 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, as the model use softmax.

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

The largest entry is the predicted class:

```
>>> np.argmax(predictions[0])
4
```

A different way to handle the labels and the loss

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)
```

The only thing this approach would change is the choice of the loss function, categorical_crossentropy **->** sparse_categorical_crossentropy

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

The importance of having sufficiently large intermediate layers

Listing 4.22 A model with an information bottleneck

```
model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(4, activation="relu"),
    layers.Dense(46, activation="softmax")
])
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))
```

64→4

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

Table of contents

Further experiments

Wrapping up

Classifying newswires: A multiclass classification example

The Reuters dataset

Loading the Reuters dataset

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

```
[ ] len(train_data)
```

```
[ ] len(test_data)
```

```
[ ] train_data[10]
```

3s completed at 3:09 PM

Exercises :D

- Try using larger or smaller layers
- Try using a single hidden layer, or three hidden layers

Predicting house prices: a regression example

The Boston Housing Price dataset:

- 404 training samples and 102 test samples with 13 features

Listing 4.23 Loading the Boston housing dataset

```
from tensorflow.keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) = (
    boston_housing.load_data())
```

Listing 4.24 Normalizing the data

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std
test_data -= mean
test_data /= std
```

Listing 4.25 Model definition

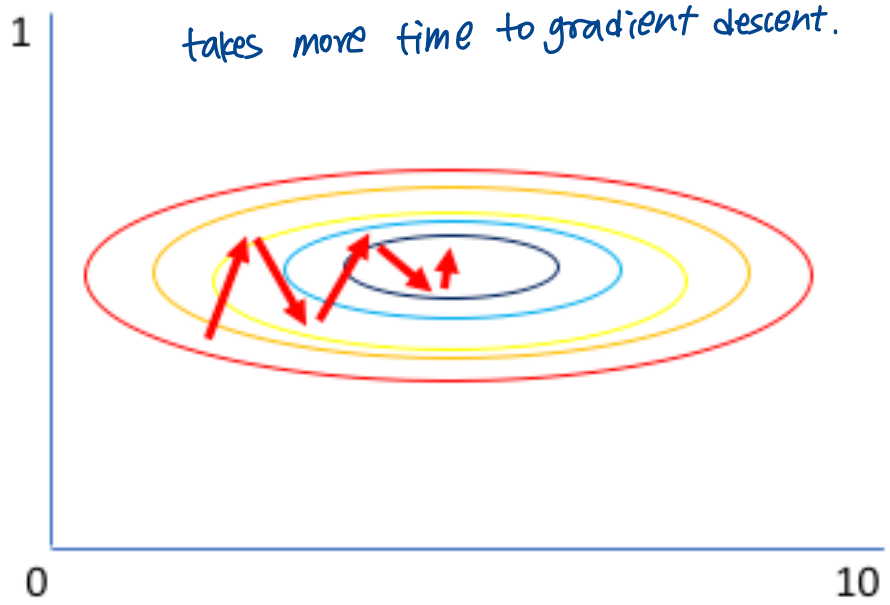
```
def build_model():
    model = keras.Sequential([
        layers.Dense(64, activation="relu"),
        layers.Dense(64, activation="relu"),
        layers.Dense(1)
    ])
    model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
    return model
```

for regression no need activation

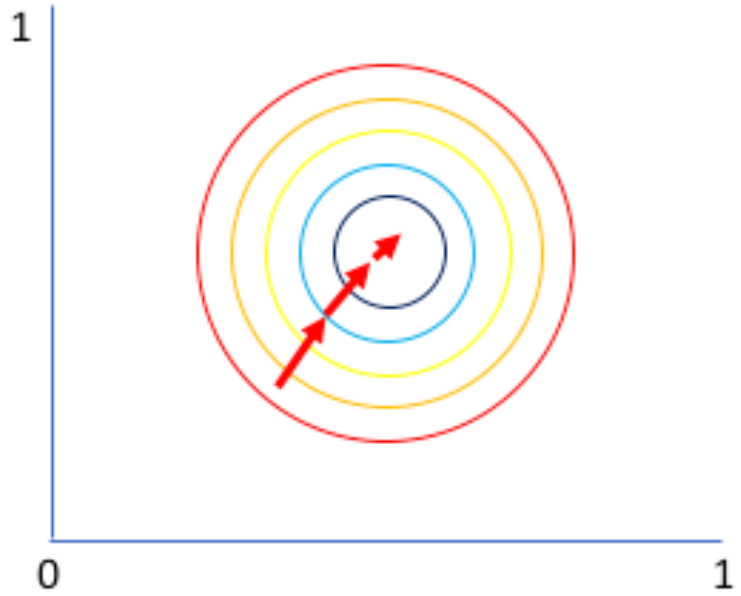
$(-\infty, \infty)$

Because we need to instantiate the same model multiple times, we use a function to construct it.

Why normalize?



Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

Validating your approach using K-fold validation

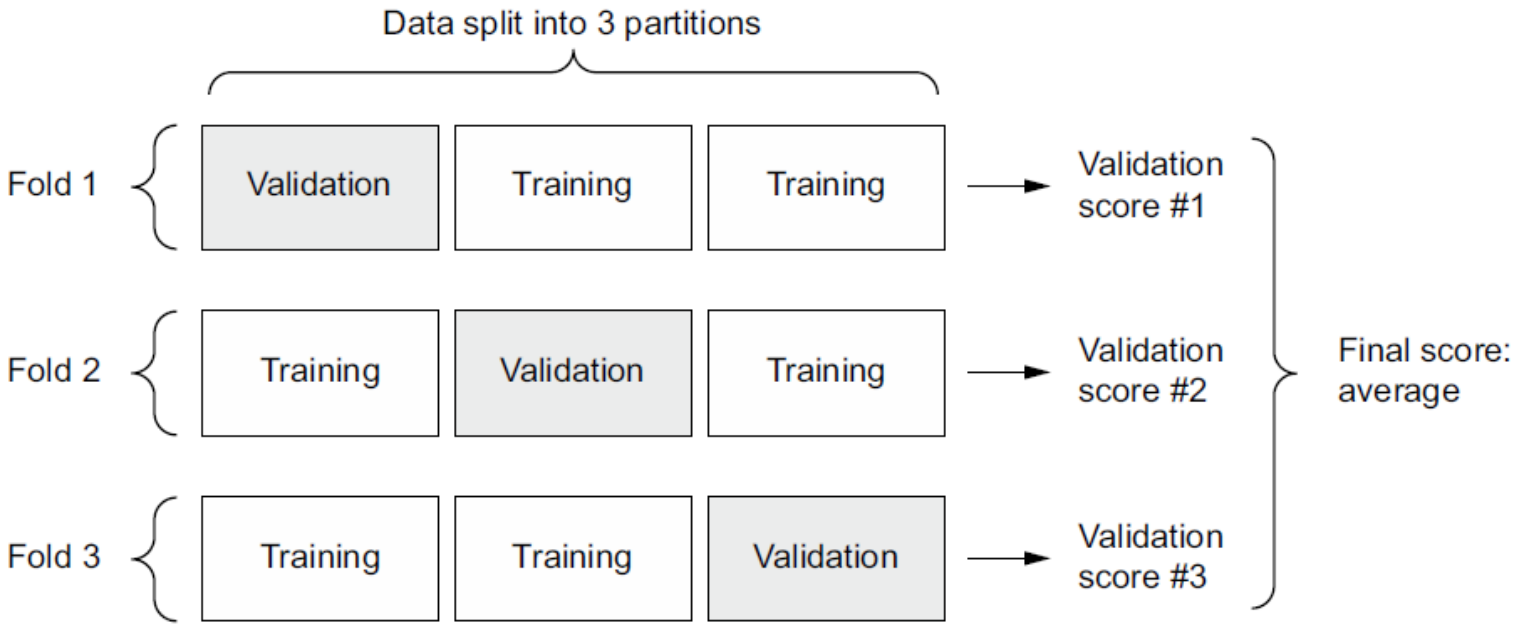


Figure 4.8 K-fold cross-validation with K=3

Listing 4.26 K-fold validation

```
k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []
for i in range(k):
    print(f"Processing fold #{i}")
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)
    model = build_model()
    model.fit(partial_train_data, partial_train_targets,
              epochs=num_epochs, batch_size=16, verbose=0)
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)
```

Prepares the validation data: data from partition #k

Prepares the training data: data from all other partitions

Builds the Keras model (already compiled)

Trains the model (in silent mode, verbose = 0)

Evaluates the model on the validation data

```
>>> all_scores
[2.112449, 3.0801501, 2.6483836, 2.4275346]
>>> np.mean(all_scores)
2.5671294
```

Let's try training the model a bit longer: 500 epochs. To keep a record of how well the model does at each epoch, we'll modify the training loop to save the per epoch validation score log for each fold.

Listing 4.27 Saving the validation logs at each fold

```
num_epochs = 500
all_mae_histories = []
for i in range(k):
    print(f"Processing fold #{i}")
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)
    model = build_model()
    history = model.fit(partial_train_data, partial_train_targets,
                        validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=16, verbose=0)
    mae_history = history.history["val_mae"]
    all_mae_histories.append(mae_history)
```

Prepares the validation data: data from partition #k

Prepares the training data: data from all other partitions

Builds the Keras model (already compiled)

Trains the model (in silent mode, verbose=0)

list of 4 components
→ each is mae_history of 500 epochs.

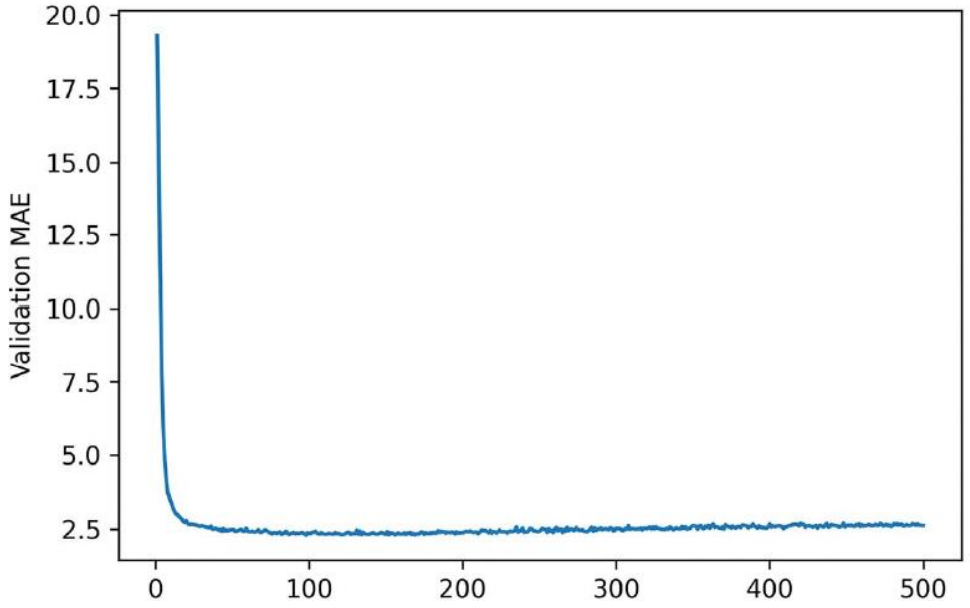
Listing 4.28 Building the history of successive mean K-fold validation scores

```
average_mae_history = [
    np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

500

Listing 4.29 Plotting validation scores

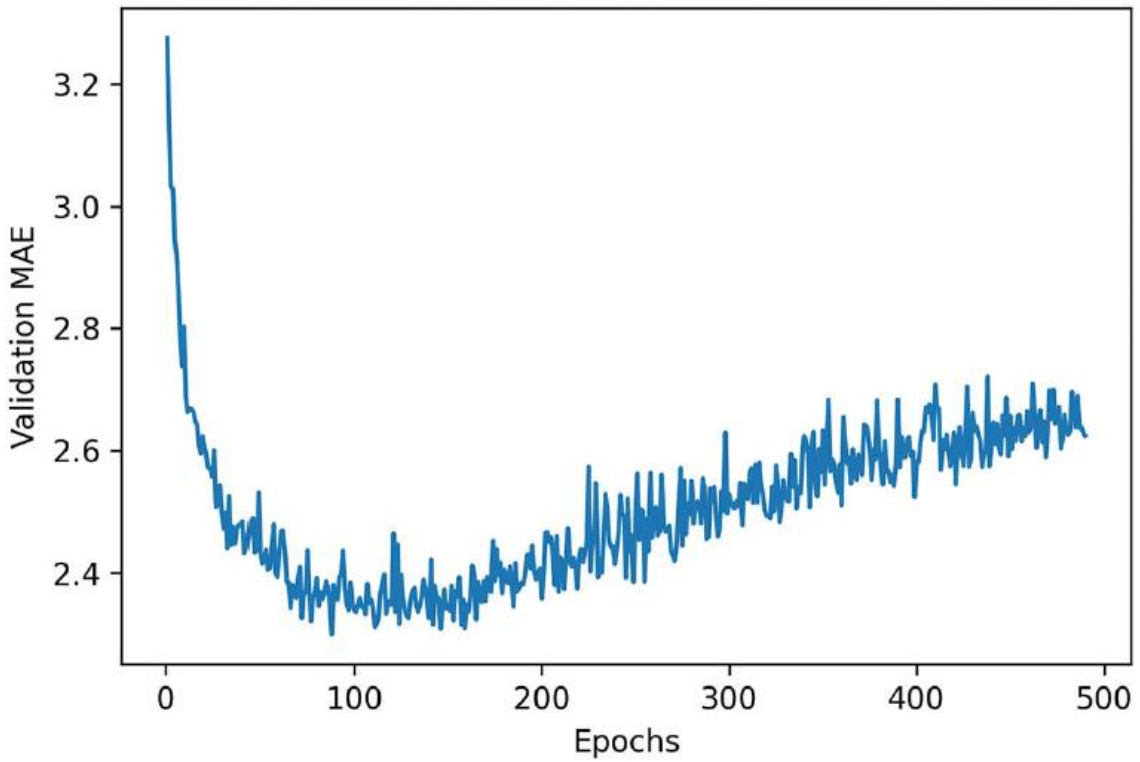
```
plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel("Epochs")
plt.ylabel("Validation MAE")
plt.show()
```



It may be a little difficult to read the plot, due to a scaling issue. Let's omit the first 10 data points.

Listing 4.30 Plotting validation scores, excluding the first 10 data points

```
truncated_mae_history = average_mae_history[10:]
plt.plot(range(1, len(truncated_mae_history) + 1), truncated_mae_history)
plt.xlabel("Epochs")
plt.ylabel("Validation MAE")
plt.show()
```



Validation MAE stops improving significantly after 120–140 epochs

Figure 4.10 Validation MAE by epoch, excluding the first 10 data points

Train a final production model on all of the training data, with the best parameters

Listing 4.31 Training the final model

Gets a fresh, compiled model

Trains it on the entirety of the data

```
model = build_model()
model.fit(train_data, train_targets,
          epochs=130, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

```
>>> test_mae_score
2.4642276763916016
```

Generating predictions on new data

```
>>> predictions = model.predict(test_data)
>>> predictions[0]
array([9.990133], dtype=float32)
```

chapter04_getting-started-with-neural-networks.i

File Edit View Insert Runtime Tools Help Cannot save changes

Share

Table of contents

+ Code + Text Copy to Drive

RAM Disk

Editing

```
Epoch 13/20
63/63 [==============================] - 1s 8ms/step - loss: 0.6067 - accuracy: 0.8454 - val_loss: 1.5636 - val_accuracy: 0.7240
Epoch 14/20
63/63 [==============================] - 1s 8ms/step - loss: 0.5687 - accuracy: 0.8520 - val_loss: 1.5497 - val_accuracy: 0.7260
Epoch 15/20
63/63 [==============================] - 0s 8ms/step - loss: 0.5406 - accuracy: 0.8594 - val_loss: 1.6150 - val_accuracy: 0.7190
Epoch 16/20
63/63 [==============================] - 1s 8ms/step - loss: 0.5133 - accuracy: 0.8690 - val_loss: 1.6685 - val_accuracy: 0.7250
Epoch 17/20
63/63 [==============================] - 0s 8ms/step - loss: 0.4922 - accuracy: 0.8752 - val_loss: 1.7528 - val_accuracy: 0.7180
Epoch 18/20
63/63 [==============================] - 1s 9ms/step - loss: 0.4682 - accuracy: 0.8820 - val_loss: 1.8296 - val_accuracy: 0.7200
Epoch 19/20
63/63 [==============================] - 1s 12ms/step - loss: 0.4489 - accuracy: 0.8849 - val_loss: 1.8731 - val_accuracy: 0.7240
Epoch 20/20
63/63 [==============================] - 1s 8ms/step - loss: 0.4308 - accuracy: 0.8898 - val_loss: 1.8995 - val_accuracy: 0.7290
<keras.callbacks.History at 0x7f3d785822d0>
```

Further experiments

Wrapping up

Predicting house prices: A regression example

The Boston Housing Price dataset

11s completed at 3:49 PM

Exercises :D

- Try varying the number of layers in the model
- Try varying the number of units per layer

Thank you!