Chapter 4:

Deep learning in practice

Outlines

1. Convolutional networks
2. Recurrent neural networks
3. Commonly used deep learning architectures
4. How to build working deep learning model from scratch
5. Project and lab

Recurrent neural networks

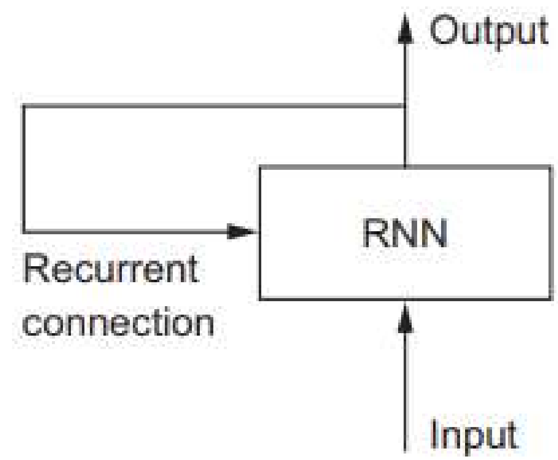
1. Support for sequences in neural networks:
   - A major characteristic of convolutional neural networks is that they have no memory.
   - Each input shown to them is processed independently, with no state kept in between inputs.
   - In order to process a sequence or a temporal series of data points, it has to show the entire sequence to the network at once: turn it into a single data point.
   - Such networks are called feedforward networks

Recurrent neural networks

1. Support for sequences in neural networks:
   - As we're reading the present sentence, we're processing it word by word while keeping memories of what came before.
   - This method gives us a fluid representation of the meaning conveyed by this sentence.
   - Biological intelligence processes information incrementally while maintaining an internal model of what it's processing.
   - New information continue to come in.

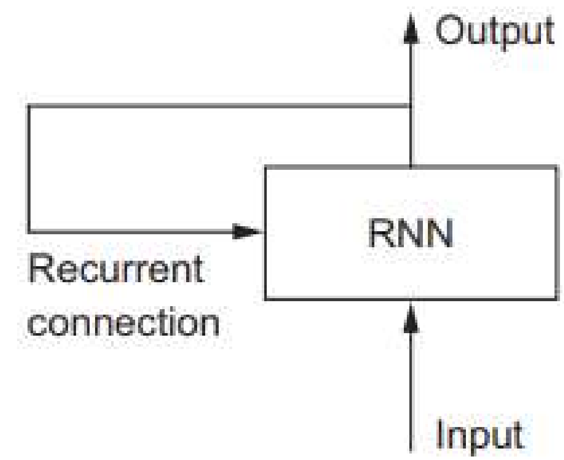
Recurrent neural networks

1. Support for sequences in neural networks:
   - A recurrent neural network (RNN) processes sequences by iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far.
   - An RNN is a type of neural network that has an internal loop.



Recurrent neural networks

1. Support for sequences in neural networks:
   - The state of the RNN is reset between processing two different.
   - One sequence, a single data point:
     - ✓ A single input to the network
   - This data point is no longer processed in a single step, but the network internally loops over sequence elements.

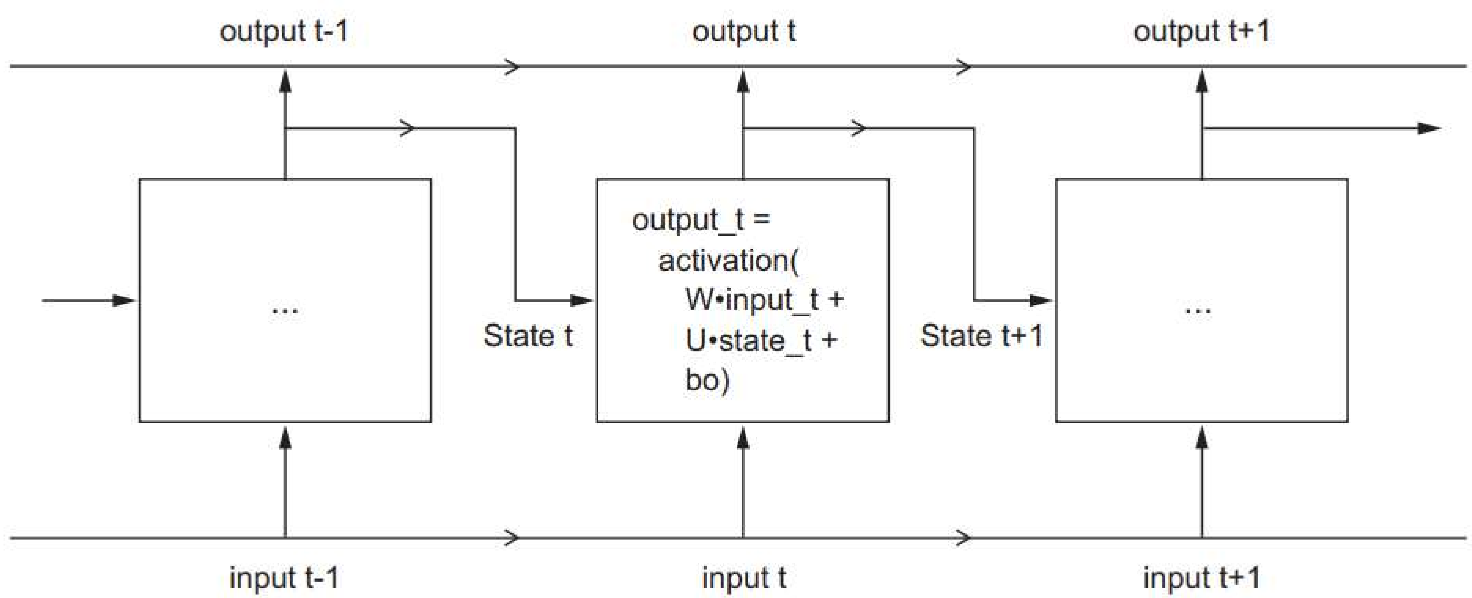


Recurrent neural networks

2. Architecture of RNN:
   - The RNN is a for loop that reuses quantities computed during the previous iteration of the loop.
   - There are many different RNNs fitting this definition.
   - RNNs are characterized by their step function.

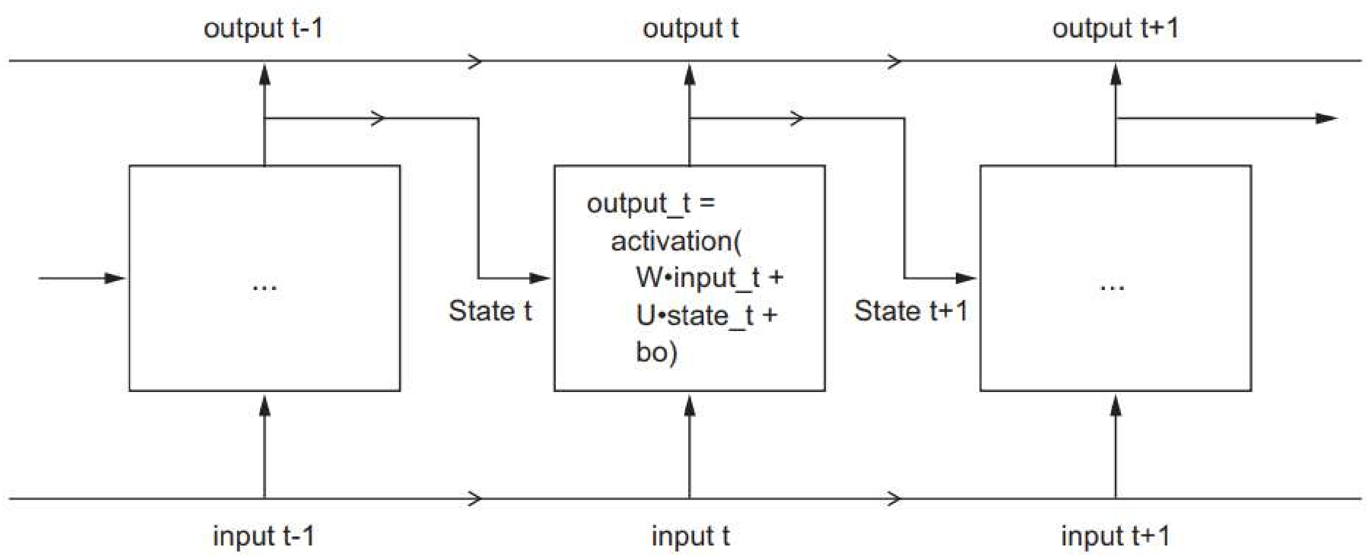
Recurrent neural networks

2. Architecture of RNN:
   - The RNN takes as input a sequence of vectors that will be encoded as a 2D tensor of size (timesteps, input_features).



Recurrent neural networks

2. Architecture of RNN:
   - The RNN loops over timesteps, and at each timestep, it considers its current state at t and the input at t (of shape (input_features,).
   - The input and current state at t will be combined to obtain the output at t.



Recurrent neural networks

2. Architecture of RNN:
   - The next step is to be this previous output.
   - For the first timestep, the previous output isn’t define - there is no current state.
   - We can initialize the state as an zero vector called the initial state of the network.
   - The final output is a 2D tensor of shape (timesteps, output_features), where each timestep is the output of the loop at time t.
   - Each timestep t in the output tensor contains information about timesteps 0 to t in the input sequence.
   - We don't need the full sequence of outputs; we just need the last output.

Recurrent neural networks

2. Architecture of RNN:

```
import numpy as np
```

```
timesteps = 100 # Number of timesteps in the input sequence
input_features = 32 # Dimensionality of the input feature space
output_features = 64 # Dimensionality of the output feature space
# Input data: random noise for the sake of the example
inputs = np.random.random((timesteps, input_features))
state_t = np.zeros((output_features,)) # Initial state: an all-zero vector
# Creates random weight matrices
W = np.random.random((output_features, input_features))
U = np.random.random((output_features, output_features))
b = np.random.random((output_features,))
successive_outputs = []
for input_t in inputs: # input_t is a vector of shape (input_features,)
    # Combines the input with the current state (the previous output)
    # to obtain the current output
    output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)
    successive_outputs.append(output_t) # Stores this output in a list
    # Updates the state of the network for the next timestep
    state_t = output_t

# The final output is a 2D tensor of shape (timesteps, output_features)
final_output_sequence = np.concatenate(successive_outputs, axis=0)
```

Recurrent neural networks

3. Keras for RNN:
   - Keras has the SimpleRNN layer to build an RNN model.

```
from keras.layers import SimpleRNN
```

   - The SimpleRNN processes batches of sequences, like all other Keras layers.
   - The SimpleRNN takes inputs of shape (batch_size, timesteps, input_features), rather than (timesteps, input_features).

Recurrent neural networks

3. Keras for RNN:
   - The SimpleRNN can be run in two different modes:
     - ✓ Return the full sequences of successive outputs for each timestep (a 3D tensor of shape (batch_size, timesteps, output_features)) .
     - ✓ Return the last output for each input sequence (a 2D tensor of shape (batch_size, output_features))

Recurrent neural networks

3. Keras for RNN:

```
from keras.models import Sequential
from keras.layers import SimpleRNN
from keras.layers import Embedding

model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32))
model.summary()
```

Recurrent neural networks

3. Keras for RNN:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 32)          320000
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 32)                2080
=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
_________________________________________________________________
```

Recurrent neural networks

3. Keras for RNN:

```
from keras.models import Sequential
from keras.layers import SimpleRNN
from keras.layers import Embedding

# returns the full state sequence:
model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32, return_sequences=True))
model.summary()
```

Recurrent neural networks

3. Keras for RNN:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 32)          320000
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, None, 32)          2080
=================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
_________________________________________________________________
```

Recurrent neural networks

3. Keras for RNN:
   - We can stack several recurrent layers one after the other in order to increase the representational power of a network.
   - In such a setup, we have to get all of the intermediate layers to return a full sequence of outputs.

```
model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32)) # Last layer only returns the last output
model.summary()
```

Recurrent neural networks

3. Keras for RNN:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 32)          320000
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, None, 32)          2080
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, 32)                2080
=================================================================
Total params: 328,320
Trainable params: 328,320
Non-trainable params: 0
_________________________________________________________________
```

Recurrent neural networks

3. Keras for RNN - IMDB:

```
from keras.datasets import imdb
from keras.models import Sequential
from keras.preprocessing import sequence
from keras.layers.embeddings import Embedding
from keras.layers import Dense, SimpleRNN
import matplotlib.pyplot as plt

max_features = 10000 # Number of words to consider as features
# Cuts off texts after this many words (among the max_features most common words)
maxlen = 500
batch_size = 32
```

Recurrent neural networks

3. Keras for RNN - IMDB:

```
# Preparing the IMDB data
print('Loading data...')
(input_train, y_train), (input_test, y_test) = imdb.load_data(
num_words=max_features)
print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')
print('Pad sequences (samples x time)')
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)
```

Recurrent neural networks

3. Keras for RNN - IMDB:

```
# Training the model with Embedding and SimpleRNN layers
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(input_train, y_train,
                    epochs=5,
                    batch_size=128,
                    validation_split=0.2)
```

Recurrent neural networks

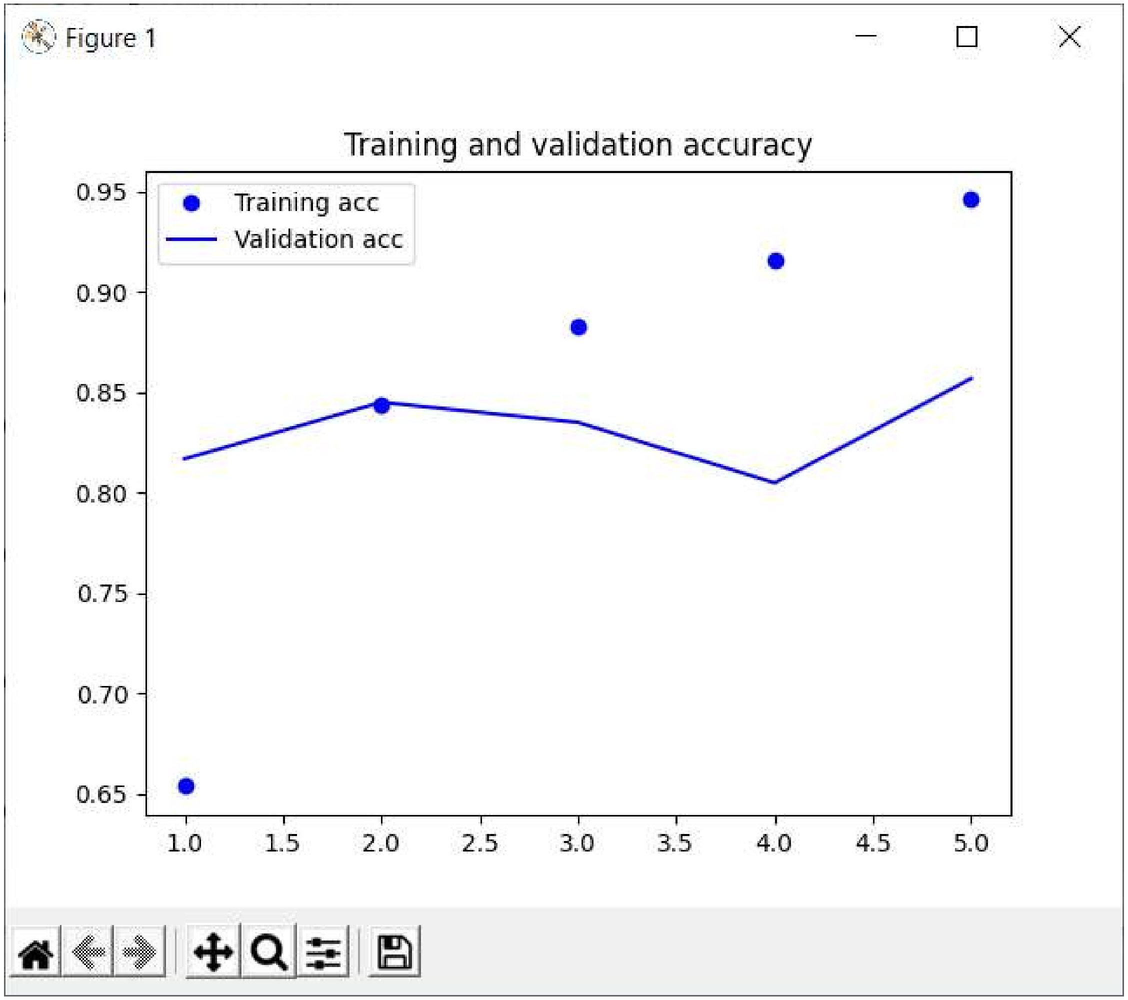
3. Keras for RNN - IMDB:

```
# Plotting results
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Recurrent neural networks

3. Keras for RNN - IMDB:



Recurrent neural networks

3. Keras for RNN - IMDB: