ipynb

June 28, 2025

[1]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, SimpleRNN,␣
 ↪LayerNormalization, MultiHeadAttention, Dropout
from tensorflow.keras.datasets import mnist
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
import numpy as np
```

[5]:
```
# Wczytanie zbioru danych IRIS
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf

# Załadowanie danych IRIS
iris = load_iris()
X_iris = iris.data
y_iris = iris.target

# Podział na dane treningowe i testowe
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.
 ↪2, random_state=42)

# Model
model_iris = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(4,)),  # Użycie Input() zamiast input_dim
    tf.keras.layers.Dense(64, activation='softplus'),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Kompilacja modelu
model_iris.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Trenowanie modelu
```

```python
model_iris.fit(X_train, y_train, epochs=50, batch_size=16,␣
 ↪validation_data=(X_test, y_test))

# Ocena modelu
loss, accuracy = model_iris.evaluate(X_test, y_test)
print(f'Accuracy for IRIS dataset: {accuracy * 100:.2f}%')
```

```
Epoch 1/50
8/8               1s 24ms/step -
accuracy: 0.2840 - loss: 1.4661 - val_accuracy: 0.2667 - val_loss: 1.1364
Epoch 2/50
8/8               0s 6ms/step -
accuracy: 0.1866 - loss: 1.0432 - val_accuracy: 0.2333 - val_loss: 0.9912
Epoch 3/50
8/8               0s 6ms/step -
accuracy: 0.4380 - loss: 0.9586 - val_accuracy: 0.5667 - val_loss: 0.9232
Epoch 4/50
8/8               0s 6ms/step -
accuracy: 0.6630 - loss: 0.9074 - val_accuracy: 0.7000 - val_loss: 0.8845
Epoch 5/50
8/8               0s 6ms/step -
accuracy: 0.6896 - loss: 0.8670 - val_accuracy: 0.7000 - val_loss: 0.8312
Epoch 6/50
8/8               0s 6ms/step -
accuracy: 0.6166 - loss: 0.8606 - val_accuracy: 0.7000 - val_loss: 0.8006
Epoch 7/50
8/8               0s 6ms/step -
accuracy: 0.7014 - loss: 0.8035 - val_accuracy: 0.8000 - val_loss: 0.7732
Epoch 8/50
8/8               0s 6ms/step -
accuracy: 0.7292 - loss: 0.7610 - val_accuracy: 0.7000 - val_loss: 0.7339
Epoch 9/50
8/8               0s 6ms/step -
accuracy: 0.6467 - loss: 0.7459 - val_accuracy: 0.7000 - val_loss: 0.7059
Epoch 10/50
8/8               0s 6ms/step -
accuracy: 0.6970 - loss: 0.7042 - val_accuracy: 0.8333 - val_loss: 0.6828
Epoch 11/50
8/8               0s 6ms/step -
accuracy: 0.7931 - loss: 0.6760 - val_accuracy: 0.8000 - val_loss: 0.6630
Epoch 12/50
8/8               0s 6ms/step -
accuracy: 0.8871 - loss: 0.6433 - val_accuracy: 0.8667 - val_loss: 0.6461
Epoch 13/50
8/8               0s 6ms/step -
accuracy: 0.8917 - loss: 0.6479 - val_accuracy: 0.8667 - val_loss: 0.6222
Epoch 14/50
8/8               0s 6ms/step -
```

```
accuracy: 0.8526 - loss: 0.6407 - val_accuracy: 0.8333 - val_loss: 0.5966
Epoch 15/50
8/8              0s 6ms/step -
accuracy: 0.7812 - loss: 0.6003 - val_accuracy: 0.8333 - val_loss: 0.5763
Epoch 16/50
8/8              0s 6ms/step -
accuracy: 0.8164 - loss: 0.5511 - val_accuracy: 0.8333 - val_loss: 0.5607
Epoch 17/50
8/8              0s 6ms/step -
accuracy: 0.8985 - loss: 0.5715 - val_accuracy: 0.9333 - val_loss: 0.5547
Epoch 18/50
8/8              0s 6ms/step -
accuracy: 0.9292 - loss: 0.5766 - val_accuracy: 0.9000 - val_loss: 0.5330
Epoch 19/50
8/8              0s 6ms/step -
accuracy: 0.8950 - loss: 0.5336 - val_accuracy: 0.8333 - val_loss: 0.5134
Epoch 20/50
8/8              0s 7ms/step -
accuracy: 0.9171 - loss: 0.5229 - val_accuracy: 0.9000 - val_loss: 0.5010
Epoch 21/50
8/8              0s 6ms/step -
accuracy: 0.9160 - loss: 0.4965 - val_accuracy: 0.9000 - val_loss: 0.4881
Epoch 22/50
8/8              0s 6ms/step -
accuracy: 0.9088 - loss: 0.4892 - val_accuracy: 1.0000 - val_loss: 0.4810
Epoch 23/50
8/8              0s 6ms/step -
accuracy: 0.9475 - loss: 0.4930 - val_accuracy: 0.9667 - val_loss: 0.4672
Epoch 24/50
8/8              0s 6ms/step -
accuracy: 0.9158 - loss: 0.4572 - val_accuracy: 0.8333 - val_loss: 0.4508
Epoch 25/50
8/8              0s 6ms/step -
accuracy: 0.8965 - loss: 0.4764 - val_accuracy: 0.9333 - val_loss: 0.4416
Epoch 26/50
8/8              0s 6ms/step -
accuracy: 0.9292 - loss: 0.4265 - val_accuracy: 1.0000 - val_loss: 0.4348
Epoch 27/50
8/8              0s 6ms/step -
accuracy: 0.9231 - loss: 0.4338 - val_accuracy: 1.0000 - val_loss: 0.4289
Epoch 28/50
8/8              0s 6ms/step -
accuracy: 0.9591 - loss: 0.4327 - val_accuracy: 1.0000 - val_loss: 0.4185
Epoch 29/50
8/8              0s 6ms/step -
accuracy: 0.9392 - loss: 0.4027 - val_accuracy: 0.8333 - val_loss: 0.4017
Epoch 30/50
8/8              0s 6ms/step -
```

```
accuracy: 0.8819 - loss: 0.4030 - val_accuracy: 0.9333 - val_loss: 0.3927
Epoch 31/50
8/8           0s 6ms/step -
accuracy: 0.9128 - loss: 0.4012 - val_accuracy: 0.9667 - val_loss: 0.3845
Epoch 32/50
8/8           0s 6ms/step -
accuracy: 0.9635 - loss: 0.3752 - val_accuracy: 1.0000 - val_loss: 0.3806
Epoch 33/50
8/8           0s 6ms/step -
accuracy: 0.9591 - loss: 0.4061 - val_accuracy: 1.0000 - val_loss: 0.3758
Epoch 34/50
8/8           0s 6ms/step -
accuracy: 0.9680 - loss: 0.3712 - val_accuracy: 0.9667 - val_loss: 0.3603
Epoch 35/50
8/8           0s 6ms/step -
accuracy: 0.9372 - loss: 0.3924 - val_accuracy: 0.9667 - val_loss: 0.3534
Epoch 36/50
8/8           0s 6ms/step -
accuracy: 0.9634 - loss: 0.3540 - val_accuracy: 1.0000 - val_loss: 0.3496
Epoch 37/50
8/8           0s 6ms/step -
accuracy: 0.9349 - loss: 0.3755 - val_accuracy: 1.0000 - val_loss: 0.3391
Epoch 38/50
8/8           0s 6ms/step -
accuracy: 0.9644 - loss: 0.3588 - val_accuracy: 1.0000 - val_loss: 0.3335
Epoch 39/50
8/8           0s 6ms/step -
accuracy: 0.9721 - loss: 0.3459 - val_accuracy: 1.0000 - val_loss: 0.3246
Epoch 40/50
8/8           0s 6ms/step -
accuracy: 0.9717 - loss: 0.3492 - val_accuracy: 1.0000 - val_loss: 0.3181
Epoch 41/50
8/8           0s 6ms/step -
accuracy: 0.9837 - loss: 0.3410 - val_accuracy: 1.0000 - val_loss: 0.3113
Epoch 42/50
8/8           0s 6ms/step -
accuracy: 0.9860 - loss: 0.3040 - val_accuracy: 1.0000 - val_loss: 0.3083
Epoch 43/50
8/8           0s 6ms/step -
accuracy: 0.9591 - loss: 0.3319 - val_accuracy: 1.0000 - val_loss: 0.3052
Epoch 44/50
8/8           0s 6ms/step -
accuracy: 0.9602 - loss: 0.3260 - val_accuracy: 1.0000 - val_loss: 0.2923
Epoch 45/50
8/8           0s 6ms/step -
accuracy: 0.9710 - loss: 0.2917 - val_accuracy: 1.0000 - val_loss: 0.2863
Epoch 46/50
8/8           0s 6ms/step -
```

```
accuracy: 0.9629 - loss: 0.2966 - val_accuracy: 1.0000 - val_loss: 0.2802
Epoch 47/50
8/8              0s 6ms/step -
accuracy: 0.9866 - loss: 0.3123 - val_accuracy: 1.0000 - val_loss: 0.2763
Epoch 48/50
8/8              0s 6ms/step -
accuracy: 0.9391 - loss: 0.2887 - val_accuracy: 1.0000 - val_loss: 0.2707
Epoch 49/50
8/8              0s 6ms/step -
accuracy: 0.9672 - loss: 0.2821 - val_accuracy: 1.0000 - val_loss: 0.2636
Epoch 50/50
8/8              0s 6ms/step -
accuracy: 0.9860 - loss: 0.2847 - val_accuracy: 1.0000 - val_loss: 0.2585
1/1              0s 22ms/step -
accuracy: 1.0000 - loss: 0.2585
Accuracy for IRIS dataset: 100.00%
```

[8]:
```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Flatten, Dense, Input
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Załadowanie zbioru MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Przekształcenie danych do formatu odpowiedniego dla sieci konwolucyjnej
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255

# Jednohotne zakodowanie etykiet
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Model konwolucyjny
model_cnn = tf.keras.models.Sequential([
    Input(shape=(28, 28, 1)),  # Użycie Input() zamiast input_shape
    Conv2D(32, kernel_size=(3, 3), activation='relu'),
    Flatten(),
    Dense(10, activation='softmax')
])

# Kompilacja modelu
model_cnn.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Trenowanie modelu
```

```
model_cnn.fit(x_train, y_train, epochs=10, batch_size=32,␣
 ↪validation_data=(x_test, y_test))

# Ocena modelu
loss, accuracy = model_cnn.evaluate(x_test, y_test)
print(f'Accuracy for MNIST dataset: {accuracy * 100:.2f}%')
```

```
Epoch 1/10
1875/1875          4s 2ms/step -
accuracy: 0.9056 - loss: 0.3274 - val_accuracy: 0.9723 - val_loss: 0.0854
Epoch 2/10
1875/1875          3s 2ms/step -
accuracy: 0.9799 - loss: 0.0667 - val_accuracy: 0.9809 - val_loss: 0.0646
Epoch 3/10
1875/1875          3s 2ms/step -
accuracy: 0.9862 - loss: 0.0463 - val_accuracy: 0.9804 - val_loss: 0.0622
Epoch 4/10
1875/1875          3s 2ms/step -
accuracy: 0.9899 - loss: 0.0343 - val_accuracy: 0.9788 - val_loss: 0.0676
Epoch 5/10
1875/1875          3s 2ms/step -
accuracy: 0.9922 - loss: 0.0259 - val_accuracy: 0.9822 - val_loss: 0.0634
Epoch 6/10
1875/1875          3s 2ms/step -
accuracy: 0.9948 - loss: 0.0181 - val_accuracy: 0.9815 - val_loss: 0.0666
Epoch 7/10
1875/1875          3s 2ms/step -
accuracy: 0.9951 - loss: 0.0159 - val_accuracy: 0.9809 - val_loss: 0.0691
Epoch 8/10
1875/1875          3s 2ms/step -
accuracy: 0.9971 - loss: 0.0100 - val_accuracy: 0.9794 - val_loss: 0.0764
Epoch 9/10
1875/1875          3s 2ms/step -
accuracy: 0.9976 - loss: 0.0092 - val_accuracy: 0.9822 - val_loss: 0.0753
Epoch 10/10
1875/1875          3s 2ms/step -
accuracy: 0.9981 - loss: 0.0066 - val_accuracy: 0.9813 - val_loss: 0.0826
313/313          0s 958us/step -
accuracy: 0.9768 - loss: 0.1003
Accuracy for MNIST dataset: 98.13%
```

[28]:
```
import tensorflow as tf
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout, Embedding, Input
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
```

```python
# Załadowanie danych IMDB, ograniczając liczbę słów do 20000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=20000)

# Przygotowanie danych (przycinanie sekwencji do 200 słów)
max_len = 200  # Maksymalna długość sekwencji
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)

# Model RNN
model_rnn = Sequential([
    Input(shape=(max_len,)),  # Wejście
    Embedding(input_dim=20000, output_dim=128),  # Warstwa embedding
    SimpleRNN(128, activation='relu'),  # Warstwa RNN
    Dropout(0.5),
    Dense(1, activation='sigmoid')  # Warstwa wyjściowa z aktywacją sigmoidalną
])

# Kompilacja modelu
model_rnn.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Trenowanie modelu
model_rnn.fit(x_train, y_train, epochs=5, batch_size=32,␣
 ↪validation_data=(x_test, y_test))

# Ocena modelu
loss, accuracy = model_rnn.evaluate(x_test, y_test)
print(f'Accuracy for IMDB sentiment analysis: {accuracy * 100:.2f}%')
```

```
Epoch 1/5
782/782            16s 18ms/step -
accuracy: 0.5814 - loss: 0.7286 - val_accuracy: 0.7285 - val_loss: 0.5396
Epoch 2/5
782/782            14s 18ms/step -
accuracy: 0.7502 - loss: 0.6388 - val_accuracy: 0.8015 - val_loss: 0.4366
Epoch 3/5
782/782            15s 19ms/step -
accuracy: 0.8242 - loss: 0.4163 - val_accuracy: 0.7933 - val_loss: 0.4453
Epoch 4/5
782/782            15s 19ms/step -
accuracy: 0.8852 - loss: 0.2963 - val_accuracy: 0.8109 - val_loss: 0.4471
Epoch 5/5
782/782            15s 19ms/step -
accuracy: 0.9117 - loss: 0.2385 - val_accuracy: 0.7648 - val_loss: 0.5535
782/782            4s 5ms/step -
accuracy: 0.7609 - loss: 0.5648
Accuracy for IMDB sentiment analysis: 76.48%
```

[32]:
```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import MultiHeadAttention, LayerNormalization,␣
 ↪Dense
from tensorflow.keras.models import Model

# Model Transformer bez pozycyjnego kodowania
class SimpleSelfAttention(tf.keras.layers.Layer):
    def __init__(self, num_heads, key_dim):
        super(SimpleSelfAttention, self).__init__()
        self.att = MultiHeadAttention(num_heads=num_heads, key_dim=key_dim)
        self.norm = LayerNormalization()

    def call(self, inputs):
        attn_output = self.att(inputs, inputs)
        return self.norm(attn_output + inputs)

# Przygotowanie danych
X_transformer = np.random.rand(100, 10, 64)  # 100 próbek, 10 timesteps, 64␣
 ↪cechy
y_transformer = np.random.rand(100, 1)  # 100 próbek, 1 wynik

# Model
inputs = tf.keras.Input(shape=(10, 64))
x = SimpleSelfAttention(num_heads=2, key_dim=64)(inputs)
x = Dense(1)(x)
model_transformer = Model(inputs=inputs, outputs=x)

# Kompilacja i trenowanie modelu
model_transformer.compile(optimizer='adam', loss='mean_squared_error')
model_transformer.fit(X_transformer, y_transformer, epochs=10, batch_size=32)

# Predykcja
y_pred_transformer = model_transformer.predict(X_transformer)
print(f'Predictions from Transformer model: {y_pred_transformer[:5]}')
```

```
Epoch 1/10
4/4              2s 6ms/step - loss:
2.2391
Epoch 2/10
4/4              0s 5ms/step - loss:
1.8231
Epoch 3/10
4/4              0s 5ms/step - loss:
1.2647
Epoch 4/10
4/4              0s 5ms/step - loss:
0.8212
```

```
Epoch 5/10
4/4             0s 5ms/step - loss:
0.6206
Epoch 6/10
4/4             0s 6ms/step - loss:
0.4570
Epoch 7/10
4/4             0s 5ms/step - loss:
0.3637
Epoch 8/10
4/4             0s 5ms/step - loss:
0.2873
Epoch 9/10
4/4             0s 6ms/step - loss:
0.2463
Epoch 10/10
4/4             0s 5ms/step - loss:
0.2285
4/4             0s 28ms/step
Predictions from Transformer model: [[[ 0.51787126]
  [ 0.8001105 ]
  [ 1.2310444 ]
  [ 0.12402429]
  [ 1.1900711 ]
  [ 0.09617489]
  [ 0.04673089]
  [ 0.52128917]
  [ 0.5379624 ]
  [ 0.39469156]]

 [[ 0.30353978]
  [ 0.89934   ]
  [ 0.14334758]
  [ 0.70972604]
  [ 0.6310268 ]
  [ 0.80318433]
  [ 0.57530934]
  [ 0.31162134]
  [ 0.61883074]
  [ 0.7406059 ]]

 [[ 0.6182496 ]
  [ 0.12775749]
  [ 0.6190339 ]
  [ 0.21829526]
  [ 0.69311154]
  [ 0.5239795 ]
  [ 0.60166913]
 [ 0.5891865 ]
 [ 0.5285214 ]
 [ 0.32452688]]

[[ 0.8504525 ]
 [ 0.44511703]
 [ 0.5371476 ]
 [ 0.9753831 ]
 [ 0.36695126]
 [-0.54100436]
 [ 0.65754807]
 [ 0.60469943]
 [ 0.20457709]
 [ 0.53949475]]

[[ 0.15623482]
 [ 0.34280673]
 [ 1.2441528 ]
 [ 0.9151012 ]
 [ 0.29407927]
 [ 0.36254665]
 [ 0.14388983]
 [ 0.7265446 ]
 [ 0.23881759]
 [ 0.05931329]]]
```