Лабораторна робота №4

Класифікація зображень засобами нейронних мереж

Мета роботи: набути практичних навичок роботи з нейронними мережами.

Література

Документація: https://www.tensorflow.org/
Документація: https://keras.io/models/sequential/
Розпізнавання об'єктів в базі даних зображень MNIST - https://www.tensorflow.org/datasets/catalog/mnist

Зміст роботи

Завдання 1. Реалізуйте роботу нейронної мережі і застосуйте її для класифікації рукописних цифр з набору даних MNIST

Набір даних складається всього з 70 000 зображень: 60 000 навчальних (використовується для створення моделі розпізнавання) і 10 000 тестових (застосовуються для оцінки точності моделі). Кожне зображення MNIST - це картинка однієї цифри, написаної від руки. Кожне зображення має розмір 28×28 пікселів. Кожне значення пікселю знаходиться в діапазоні від 0 (білий колір) до 255 (чорний колір). Проміжні значення відображають відтінки сірого. Зображення в бінарному вигляді записані в один файл.

Етапи розробки:

Завдання рекомендується виконувати в Google Colaboratory, де вже встановлено TensorFlow і багато інших корисних бібліотек. Бібліотеки, що будуть використані:

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

В бібліотеці Keras реалізовані методи для завантаження популярних наборів даних для алгоритмів машинного навчання *mnist.load_data()* - повертає два набора даних для навчання і для тестів:

```
mnist_dataset = tf.keras.datasets.mnist.load_data()

(x_train, y_train), (x_test, y_test) = mnist_dataset
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

Проводимо візуалізацію завантаженого набору даних:

```
def visualize_dataset_item(x, y):
  print("Цифра:", y)
```

```
  plt.imshow(x, cmap=plt.cm.gray)
  plt.axis('off')

item_index = 0
visualize_dataset_item(x_train[item_index], y_train[item_index])
```

Далі можна побудувати гістограми (діаграми розподілу) за допомогою функції **hist()**:

```
plt.hist(y_train, bins = 10)
plt.show()
plt.hist(y_test, bins = 10)
plt.show()
```

Проводимо стандартизацію зображень:

```
def preprocess_image(image):
  image = tf.reshape(image, [28, 28, 1])
  image = tf.image.per_image_standardization(image)

  return image
```

Перед тим, як почати навчання, потрібно провести попередню обробку вхідних даних.

```
total_count = len(y_train)

images_dataset = tf.data.Dataset.from_tensor_slices(x_train)
images_dataset = images_dataset.map(preprocess_image)

labels_dataset = tf.data.Dataset.from_tensor_slices(y_train)

dataset = tf.data.Dataset.zip((images_dataset, labels_dataset))
dataset = dataset.shuffle(buffer_size=1024)

train_dataset = dataset.take(round(total_count * 0.8))
val_dataset = dataset.skip(round(total_count * 0.8))

train_dataset = train_dataset.batch(128, drop_remainder = False)
train_dataset = train_dataset.prefetch(4)

val_dataset = val_dataset.batch(1, drop_remainder = False)
val_dataset = val_dataset.prefetch(4)
```

Процес навчання штучної нейронної мережі розглядається як налаштування архітектури і ваги зв'язків між нейронами (параметри) для ефективного виконання поставлених перед мережею завдань. Існує два великих класи навчання: клас детермінованих методів і клас стохастичних методів.

До класу детермінованих методів входять методи, в основі яких лежить ітеративна корекція параметрів мережі, в ході поточної ітерації, яка ґрунтується на

поточних параметрах. Основним і найпоширенішим методом навчання мереж сьогодні є метод зворотного поширення помилки.

До класу стохастичних методів входять методи, що змінюють параметри мережі випадковим чином і зберігають тільки ті зміни параметрів, які призвели до поліпшення результатів роботи. Стохастичні алгоритми навчання реалізуються за допомогою порівняння помилок.

Групування набору шарів у файл (модель нейронної мережі) і проведення навчання:

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), input_shape=(28, 28, 1), activation='
relu'),
  tf.keras.layers.MaxPool2D(),
  tf.keras.layers.Conv2D(64, (2, 2), activation='relu'),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy'
, metrics=['sparse_categorical_accuracy'],)
```

Для навчання мережі використаємо метод fit

```
history = model.fit(train_dataset, epochs = 5, validation_data = val_datas
et)
```

В коді використано міра помилки – sparse categorical crossentropy, метрика оптимізації adam (метод адаптивної інерції), який поєднує в собі і ідею накопичення руху і ідею слабшого поновлення ваги для типових ознак. Можна також використати метод accuracy.

Побудуємо графік історії навчання:

```
def show_history_charts(histpry):
  plt.plot(history.history['loss'])
  plt.plot(history.history['val_loss'])
  plt.title('model loss')
  plt.ylabel('loss')
  plt.xlabel('epoch')
  plt.legend(['train', 'validation'], loc='upper left')
  plt.show()

  plt.plot(history.history['sparse_categorical_accuracy'])
  plt.plot(history.history['val_sparse_categorical_accuracy'])
  plt.title('model sparse categorucal accuracy')
  plt.ylabel('accuracy')
  plt.xlabel('epoch')
  plt.legend(['train', 'validation'], loc='upper left')
  plt.ylim(0.9, 1)
  plt.show()
```

```
show_history_charts(history)
```

Отримані значення свідчать про *overfitting*. Спробуємо змінити модель додавши додатковий шар Dropout і змінити кількість ітерацій навчання:

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3, 3), input_shape=(28, 28, 1), activation='
relu'),
  tf.keras.layers.MaxPool2D(),
  tf.keras.layers.Conv2D(64, (2, 2), activation='relu'),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.35),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy'
, metrics=['sparse_categorical_accuracy'],)

history = model.fit(train_dataset, epochs = 10, validation_data = val_data
set)
```

Після проведених змін і перенавчення побудуємо графік історії навчання. Проаналізуємо отримані дані.

За даними графіка можна побачити, що ситуація з перенавчанням значно покращилася, але точність прогнозування моделі зазнала лише незначного покращення. Спробуємо змінити архітектуру моделі та кількість ітерацій навчання.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (5, 5), padding = 'same', input_shape=(28, 28
, 1), activation='relu'),
  tf.keras.layers.Conv2D(32, (5, 5), padding = 'same', activation='relu'),
  tf.keras.layers.MaxPool2D(),
  tf.keras.layers.Conv2D(64, (2, 2), padding = 'same', activation='relu'),
  tf.keras.layers.Conv2D(64, (2, 2), padding = 'same', activation='relu'),
  tf.keras.layers.MaxPool2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.35),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy'
, metrics=['sparse_categorical_accuracy'],)

history = model.fit(train_dataset, epochs = 20, validation_data = val_data
set)
```

Після проведених маніпуляцій з моделлю побудуємо графік історії навчання. Проведемо аналіз отриманих даних. Якщо дані влаштовують, проведемо тестування отриманої моделі:

```
total_count = len(y_test)

test_images_dataset = tf.data.Dataset.from_tensor_slices(x_test)
test_images_dataset = test_images_dataset.map(preprocess_image)

test_labels_dataset = tf.data.Dataset.from_tensor_slices(y_test)

test_dataset = tf.data.Dataset.zip((test_images_dataset, test_labels_datas
et))

test_dataset = test_dataset.batch(128, drop_remainder = False)
test_dataset = test_dataset.prefetch(4)

y_pred = model.predict(test_dataset)
```

Проведемо візуалізацію даних

```
item_index = 3
visualize_dataset_item(x_test[item_index], np.argmax(y_pred[item_index]))
```

Можна переглянути зображення, які мережа класифікує не коректно:

```
invalid_prediction_indexes = []
for i in range(len(y_test)):
  if np.argmax(y_pred[i]) != y_test[i]:
    invalid_prediction_indexes.append(i)


index = 10

print("Predicted:", np.argmax(y_pred[invalid_prediction_indexes[index]]))
visualize_dataset_item(x_test[invalid_prediction_indexes[index]], y_test[i
nvalid_prediction_indexes[index]])
```

Оцінимо якість навчання мережі на тестових даних:

```
print("Test accuracy:", str(100 -
 round(len(invalid_prediction_indexes) / len(y_pred) * 100, 2)) + "%")
```

Завдання 2. Провести експерименти з архітектурою нейронної мережі для досягнення більшого відсотку точності. Зберегти модель нейронної мережі для подальшого її використання.

Завдання 3. Провести тестування отриманої моделі на незалежних даних.

Контрольні запитання

1. Назвіть сфери застосування нейронних мереж.
2. Назвіть види нейронних мереж, що класифікуються за топологією.
3. До яких мереж належать мережі без зворотних зв’язків?
4. На які види поділяють нейронні мережі за принципом структури нейронів?
5. Поясніть поняття “Перцептрон”?
6. З яких типів елементів складається Перцептрон?
7. Який Перцептрон називається багатошаровим?
8. Які шари використовуються в CNN?
9. Назвіть та охарактеризуйте етапи побудови нейромережевої моделі.
10. Яким чином перевіряється правильність роботи навченої мережі?
11. Охарактеризуйте вид навчання нейронної мережі “навчання з вчителем”.
12. Охарактеризуйте вид навчання нейронної мережі “навчання без вчителя”.
13. Назвіть недоліки алгоритму зворотного розповсюдження помилки.
14. Охарактеризуйте метод градієнтного спуску (gradient descent).
15. Охарактеризуйте набор даних MNIST