Metody obliczeniowe optymalizacji

Optymalizatory sieci neuronowych

Poniżej podano kod w jezyku Python, który korzystając z modułu Tensorflow oraz podmodułu Keras, buduje architekturę konwolucyjnej sieci neuronowej (CNN) i trenuje model przy użyciu optymalizatora SGD (Stochastic Gradient Descent, Stochastyczny Spadek Gradientu). Z dzisiejszego punktu widzenia jest to raczej sieć płytka (nie jest to więc deep learning), biorąc pod uwagę liczbę warstw. Sieć jest trenowana na zbiorze Fashiom-MNIST, którego charakterystyka jest następująca:

- Rozmiar obrazów: 28x28 pikseli
- Głębia koloru: skala szarości (1 kanał)
- Liczba obrazów: 70 000
  - Zbiór treningowy: 60 000
  - Zbiór walidacyjny: 10 000
- Liczba klas: 10
- Etykiety klas:
  0. koszulka (T-shirt)
  1. spodnie
  2. sweter
  3. sukienka
  4. płaszcz
  5. sandał
  6. koszula
  7. sneaker
  8. torba
  9. trzewik (botka)

W każdym zadaniu optymalizacyjnym musi być ustalona jakaś funkcja celu. W naszym wypadku jest to tzw. funkcja straty, którą możemy zapisać jako:

$$-\ln(p) \rightarrow min,$$

gdzie p jest prawdopodobieństwem, jakie sieć przypisała prawidłowej klasie (tej, która powinna być wynikiem klasyfikacji pod warunkiem, że klasyfikator się nie pomylił). Taka funkcja straty nazywa się entropią krzyżową (ang. Cross-Entropy Loss). Wartość p sieć oblicza na podstawie jej parametrów (wagi i przesunięcia (ang. bias) neuronów), a więc to te parametry można uznać za zmienne decyzyjne optymalizacji.

Zadaniem studenta jest zmodyfikowanie kodu (miejsca oznaczone komentarzem [TODO]), tak aby uwzględniał również optymalizatory:

- **SGD** z parametrem *momentum* (pęd, symulacja efektu bezwładności) ustawionym na 0.9;
- **Adagrad** (Adaptive Gradient Algorithm, Adaptacyjny Algorytm Gradientowy);
- **RMSprop** (Root Mean Square Propagation, Propagacja Pierwiastka Średniej Kwadratowej);
- **Adam** (Adaptive Moment Estimation).

Następnie należy przeanalizować i porównać wykresy straty oraz dokładności obliczonych dla zbioru walidacyjnego i każdego z pięciu optymalizatorów w kolejnych epokach treningu. Należy zwrócić uwagę, że strata nie jest tym samym co dokładność, co można zaobserwować porównując wykresy. Dokładność bierze

pod uwagę jedynie to, czy klasa z najwyższym prawdopodobieństwem jest prawidłowa, natomiast strata uwzględnia również stopień pewności modelu dla klasyfikacji poprawnych oraz niepoprawnych.

Po przeanalizowaniu wykresów należy poszukać informacji na temat wykorzystanych optymalizatorów.

Oprócz wymienionych pięciu optymalizatorów, w Keras są dostępne również inne, bardziej zaawansowane i nowoczesne optymalizatory. Omawianie ich działania wykracza poza temat zajęć, natomiast warto w skrócie wiedzieć w jakich przypadkach zalecane jest ich użycie:

- **AdamW** – do trenowania nowoczesnych sieci typu Transformer, dużych modeli językowych;
- **FTRL** – do tworzenia systemów rekomendacyjnych z rzadkimi danymi (ang. sparse data – dane, w których występuje dużo zer);
- **Nadam** – kiedy zależy nam na maksymalnej szybkości uzyskania zbieżności;
- **Adadelta** – kiedy nie chcemy zajmować się dobieraniem współczynnika uczenia (ang. learning rate), ponieważ jest on dopierany automatycznie;
- **Lion** – kiedy mamy bardzo mało pamięci VRAM w GPU, a chcemy wytrenować stosunkowo duży model.

Kod:

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

gpus = tf.config.list_physical_devices('GPU')
if gpus:
    print(f"Znaleziono GPU: {len(gpus)}.")
else:
    print("Brak GPU. Keras użyje CPU.")

# hiperparametry
EPOCHS = 10
BATCH_SIZE = 128
LEARNING_RATE = 0.01

# architektura sieci
def build_fashion_cnn():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(28,28,1)),
        tf.keras.layers.Conv2D(16,kernel_size=(3,3),padding='same',activation='relu'),
        tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
        tf.keras.layers.Conv2D(32,kernel_size=(3,3),padding='same',activation='relu'),
        tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128,activation='relu'),
        tf.keras.layers.Dense(10,activation='softmax') # wyjście dla 10 klas ubrań
    ])
    return model

def build_bean_mlp(input_dim,num_classes):
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(input_dim,)),
        tf.keras.layers.Dense(128,activation='relu'),
        tf.keras.layers.Dense(64,activation='relu'),
        tf.keras.layers.Dense(num_classes,activation='softmax')
    ])
    return model

# pobranie zbioru danych
def get_fashion_mnist():
    (X_train,y_train),(X_test,y_test) = tf.keras.datasets.fashion_mnist.load_data()
    # normalizacja do przedziału 0-1 i dodanie wymiaru kanału (wymagane przez Conv2D)
    X_train = np.expand_dims(X_train,-1).astype('float32') / 255.0
```

```
    X_test = np.expand_dims(X_test,-1).astype('float32') / 255.0
    return (X_train,y_train),(X_test,y_test)

if __name__ == "__main__":
    # słownik z obiektami optymalizatorów
    def get_optimizer(name):
        optimizers_dict = {
            'SGD':tf.keras.optimizers.SGD(learning_rate=LEARNING_RATE)
            # [TODO] dopisać kolejne optymalizatory
        }
        return optimizers_dict[name]

    # [TODO] dopisać nazwy kolejnych optymalizatorów
    optimizers_to_test = ['SGD']
    all_results = {}

    # wczytanie zbioru danych Fashion-MNIST (Obrazy)
    print("Uruchamianie eksperymentu na Fashion-MNIST.")
    (X_train,y_train),(X_test,y_test) = get_fashion_mnist()

    for opt_name in optimizers_to_test:
        print(f"\nTestowanie optymalizatora: {opt_name}")

        # budowa nowej sieci
        model = build_fashion_cnn()

        # przypisanie optymalizatora i funkcji straty
        model.compile(optimizer=get_optimizer(opt_name),
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy']
                      )

        # uczenie
        history = model.fit(
            X_train,y_train,
            batch_size=BATCH_SIZE,
            epochs=EPOCHS,
            validation_data=(X_test,y_test),
            verbose=1
        )
        all_results[opt_name] = history.history

    # wizualizacja wyników
    plt.figure(figsize=(14,6))

    # wykres straty na zbiorze walidacyjnym
    plt.subplot(1,2,1)
    for name,hist in all_results.items():
        plt.plot(range(1,EPOCHS + 1),hist['val_loss'],
                 marker='o',label=name)
    plt.title('Funkcja straty (Validation Loss)')
    plt.xlabel('Epoka')
    plt.ylabel('Loss')
    plt.grid(True)
    plt.legend()

    # wykres dokładności na zbiorze walidacyjnym
    plt.subplot(1,2,2)
    for name,hist in all_results.items():
        plt.plot(range(1,EPOCHS + 1),hist['val_accuracy'],marker='s',label=name)
    plt.title('Dokładność (Validation Accuracy) %')
    plt.xlabel('Epoka')
    plt.ylabel('Accuracy')
    plt.grid(True)
    plt.legend()

    plt.tight_layout()
    plt.show()
```