Laboratorium: Strojenie hiperparametrów

June 2, 2022

1 Zakres ćwiczeń

- Zbadanie wpływu poszczególnych hiperparametrów na proces uczenia i jakość modelu.
- Zaobserwowanie niekorzystnych efektów w procesie uczenia (np. niestabilność gradientów).
- Zbadanie różnic w działaniu algorytmów optymalizacji innych niż SGD.
- Automatyzacja poszukiwania optymalnych hiperparametrów przy pomocy wrappera scikeras i narzędzi optymalizacyjnych scikit-learn.

2 Zadania

2.1 Poszukiwania ręczne

Pobierz zestaw danych Boston Housing:

```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.boston_housing.
↪load_data()
```

Przygotuj funkcję:

```
def build_model(n_hidden, n_neurons, optimizer, learning_rate, momentum=0):
    model = tf.keras.models.Sequential()
    # ...
    return model
```

budującą model według parametrów podanych jako argumenty:

- `n_hidden` – liczba warstw ukrytych,
- `n_neurons` – liczba neuronów na każdej z warstw ukrytych,
- `optimizer` – gradientowy algorytm optymalizacji, funkcja powinna rozumieć wartości: `sgd`, `nesterov`, `momentum` oraz `adam`,
- `learning_rate` – krok uczenia,
- `momentum` – współczynnik przyspieszenia dla algorytmów z pędem.

Model powinien być odpowiedni dla regresji wartości zgodnie ze zbiorem danych. Zwracany model powinien być skompilowany z wykorzystaniem błędu średniokwadratowego (MSE) jako funkcji celu oraz błędu średniego (MAE) jako dodatkowej metryki.

Jako bazową przyjmij konfigurację sieci:

- 1 warstwa ukryta,
- krok uczenia: 10^{-5},

- liczba neuronów na warstwię: 25,
- algorytm optymalizacji: SGD.

Przeprowadź 5 eksperymentów. W każdym zmieniany będzie jeden parametr (względem wartości domyślnych):

1. krok uczenia (`lr`): 10^{-6}, 10^{-5}, 10^{-4},
2. liczba warstw ukrytych (`hl`): od 0 do 3,
3. liczba neuronów na warstwę (`nn`): 5, 25, 125,
4. algorytm optymalizacji (`opt`): wszystkie 4 algorytmy (pęd = 0.5).
5. pęd (`mom`): 0.1, 0.5, 0.9 (dla algorytmu `momentum`).

Przy uczeniu wykorzystaj mechanizm *early stopping* o cierpliwości równej 10 i minimalnej poprawie funkcji straty równej 1.00, uczenie maksymalnie przez 100 epok.

Każdy przebieg powinien zbierać dane dla TensorBoard, w katalogu bazowym `tb_logs` będącym podkatalogiem bieżącego katalogu, w podkatalogach o nazwie:

`UNIXTIME_NAME_VALUE (np. 1654154941_lr_0.001)`

gdzie `UNIXTIME` to czas (UNIX epoch, sekundowy) w chwili rozpoczęcia danego eksperymentu, `NAME` to nazwa eksperymentu podana powyżej, `VALUE` to wartość zmienianego parametru.

Aktualny czas UNIX łatwo pobrać np. przy pomocy polecenia:

```
ts = int(time.time())
```

Dodatkowo, wyniki dla każdego z eksperymentów zapisz w pliku o nazwie `NAME.pkl` (gdzie `NAME` to nazwa eksperymentu, np. `lr.pkl`), w postaci zapiklowanej listy 3-elementowych krotek (`VALUE`, `mse`, `mae`), np. dla kroku uczenia:

```
[(0.01, 89.29692840576172, 6.8168745040893555),
 (0.001, 60608.32421875, 245.99798583984375),
 (0.0001, 379.87762451171875, 17.05248260498047)]
```

Przed eksperymentami wyczyść sesję TensorFlow i ustal generatory liczb losowych:

```
tf.keras.backend.clear_session()
np.random.seed(42)
tf.random.set_seed(42)
```

Po każdym z eksperymentów przeanalizuj jego przebieg przy pomocy TensorBoard.

10 pkt.

2.2 Automatyczne przeszukiwanie przestrzeni hiperparametrów

W tym ćwiczeniu wykorzystamy narzędzie `RandomizedSearchCV` pakietu scikit-learn.

Aby móc go użyć, należy nasz model obudować wrapperem scikeras.

Przygotuj słownik zawierający przeszukiwane wartości parametrów:

```
param_distribs = {
    "model__n_hidden": [...],
    "model__n_neurons": [...],
```

```
    "model__learning_rate": [...],
    "model__optimizer": [...],
    "model__momentum": [...]
}
```

Przygotuj callback *early stopping* i obuduj przygotowaną wcześniej funkcję `build_model` obiektem `KerasRegressor`:

```
import scikeras
from scikeras.wrappers import KerasRegressor

es = tf.keras.callbacks.EarlyStopping(patience=10, min_delta=1.0, verbose=1)

keras_reg = KerasRegressor(build_model, callbacks=[es])
```

Przygotuj obiekt `RandomizedSearchCV`, tak aby wykonał 30 iteracji przy 3-krotnej walidacji krzyżowej, a następnie przeprowadź uczenie:

```
from sklearn.model_selection import RandomizedSearchCV

rnd_search_cv = RandomizedSearchCV(keras_reg,
                                   param_distribs,
                                   n_iter=20,
                                   cv=3,
                                   verbose=2)

rnd_search_cv.fit(X_train, y_train, epochs=100, validation_split=0.1)
```

Zapisz najlepsze znalezione parametry w postaci słownika do pliku `rnd_search.pkl`.

6 pkt.