Laboratorium: Strojenie hiperparametrów

May 21, 2025

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,
  - pakietu KerasTuner.

2 Zadania

2.1 Pobieranie danych

Pobierz zestaw danych California Housing i dokonaj jego podziału oraz normalizacji:

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

housing = fetch_california_housing()

X_train_full, X_test, y_train_full, y_test = train_test_split(housing.data,␣
 ↪housing.target, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full,␣
 ↪y_train_full, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

2.2 Przeszukiwanie przestrzeni hiperparametrów przy pomocy scikit-learn

Celem ćwiczenia jest przejrzenie przestrzeni parametrów w następujących zakresach:

1. krok uczenia: $[3 \cdot 10^{-4}, 3 \cdot 10^{-2}]$,
2. liczba warstw ukrytych: od 0 do 3,
3. liczba neuronów na warstwę: od 1 do 100,
4. algorytm optymalizacji: `adam`, `sgd` lub `nesterov`.

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

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

Uwaga: scikit-learn od wersji 1.6.0 zmienił API, co może powodować błąd przy korzystaniu z scikeras. Dla pewności użyj wersji 1.5.2.

Przygotuj słownik zawierający przeszukiwane wartości parametrów:

```
param_distribs = {
    "model__n_hidden": ...,
    "model__n_neurons": ...,
    "model__learning_rate": ...,
    "model__optimizer": ...
}
```

Listę wartości reprezentujących rozkład dla kroku uczenia możesz uzyskać np. przy pomocy tej funkcji pakietu SciPy:

```
from scipy.stats import reciprocal
reciprocal(3e-4, 3e-2).rvs(1000).tolist()
```

Przygotuj funkcję:

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

Spróbuj uruchomić ręcznie funkcję, zmieniając tylko jeden z parametrów – na przykład krok uczenia w zakresie od `1e-6` do `1e-1`.

Przeprowadź trening sieci, np. przez 40 epok, a następnie zwizualizuj wyniki, np. tak (zakładając że zmienne `h1`…`h6` zawierają historię treningów):

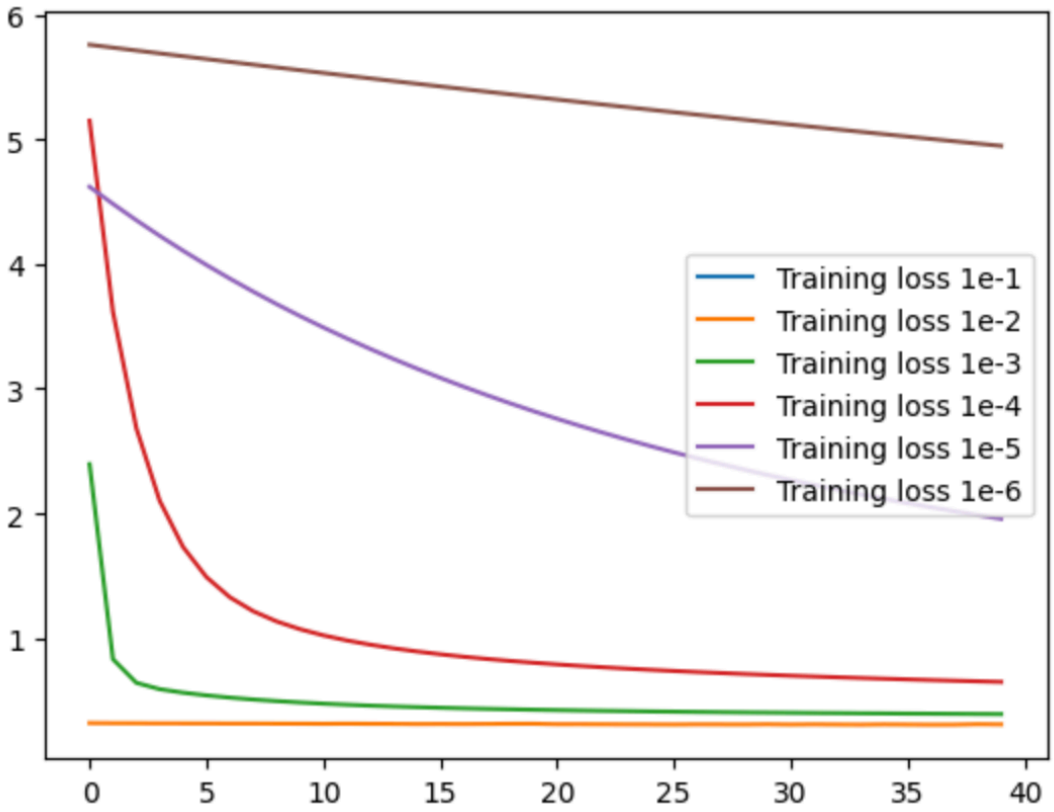
```
h = [h1, h2, h3, h4, h5, h6]
for i, h_i in enumerate(h):
  plt.plot(h_i.history['loss'], label=f"Training loss 1e-{i+1}")
plt.legend()
```

Spróbuj uruchomić ręcznie funkcję, zmieniając tylko jeden z parametrów – na przykład krok uczenia w zakresie od `1e-6` do `1e-1`.

Przeprowadź trening sieci, np. przez 40 epok, a następnie zwizualizuj wyniki, np. tak (zakładając że zmienne `h1`…`h6` zawierają historię treningów):

```
h = [h1, h2, h3, h4, h5, h6]
for i, h_i in enumerate(h):
  plt.plot(h_i.history['loss'], label=f"Training loss 1e-{i+1}")
plt.legend()
```

Wynik powinien wyglądać mniej więcej tak:



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 5 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=5,
                                    cv=3,
                                    verbose=2)

rnd_search_cv.fit(X_train, y_train, epochs=100, validation_data=(X_valid,␣
 ↪y_valid), verbose=0)
```

Zapisz najlepsze znalezione parametry w postaci słownika do pliku `rnd_search_params.pkl` w postaci słownika o następującej strukturze:

```
{'model__optimizer': 'adam',
 'model__n_neurons': 42,
 'model__n_hidden': 3,
 'model__learning_rate': 0.004003820130936959}
```

4 pkt.

Zapisz obiekt `RandomizedSearchCV` do pliku `rnd_search_scikeras.pkl`.

6 pkt.

2.3 Przeszukiwanie przestrzeni hiperparametrów przy pomocy Keras Tuner

Przeprowadź podobny eksperyment przy pomocy KerasTuner. Przyjmij identyczne jak w poprzednim ćwiczeniu zakresy hiperparametrów.

Przygotuj funkcję `build_model_kt`, przyjmującą obiekt HyperParameters jako wejście. Powinna ona w pierwszej części definiować hiperparametry, a w drugiej – przeprowadzić budowę modelu:

```
import keras_tuner as kt

def build_model_kt(hp):
    n_hidden = hp.Int("n_hidden", min_value=0, max_value=3, default=2)
    # (...)

    model = tf.keras.models.Sequential()
    # (...)

    # model.compile(...)
    return model
```

Przygotuj wybrany *tuner* spośród dostępnych w Keras Tuner, np.:

```
random_search_tuner = kt.RandomSearch(
    build_model, objective="val_mse", max_trials=10, overwrite=True,
    directory="my_california_housing", project_name="my_rnd_search", seed=42)
```

Przygotuj również callback TensorBoard do zbierania danych w podkatalogu `tensorboard` w katalogu projektu:

```
root_logdir = os.path.join(random_search_tuner.project_dir, 'tensorboard')
tb = tf.keras.callbacks.TensorBoard(root_logdir)
```

Uruchom przeszukiwanie dla maksymalnie 100 epok na próbę. Pamiętaj o podaniu danych walidacyjnych (`X_valid`, `y_valid`) oraz utworzonego przed chwilą callbacku TensorBoard oraz stworzonego wcześniej callbacku *early stopping*.

Uruchom TensorBoard i przeanalizuj proces strojenia hiperparametrów w zakładce *HPARAMS*:

```
%load_ext tensorboard
%tensorboard --logdir {root_logdir}
```

Zapisz do pliku `kt_search_params.pkl` parametry najlepszego znalezionego modelu w postaci słownika, np.:

```
{'n_hidden': 3,
 'n_neurons': 45,
 'learning_rate': 0.0008960175671873151,
 'optimizer': 'adam'}
```

4 pkt.

Zapisz najlepszy uzyskany model do pliku `kt_best_model.keras`.

6 pkt.

3 Prześlij rozwiązanie

Skrypt wykonujący powyższe zadania umieść w swoim repozytorium, w pliku `lab10/lab10.py`.