Uczenie Maszynowe
Laboratorium 7: Restricted Boltzmann Machines i Deep Belief Networks

1 Cele laboratorium

- Zastosowanie Restricted Boltzmann Machines (RBM) oraz Deep Belief Networks (DBN) do ekstrakcji cech obrazu
- Rekonstrukcja i generacja nowych danych za pomocą RBM
- Błąd rekonstrukcji

2 Literatura

- Wykład: *Modele Generatywne 1, 2, 3*
- Amir Ali, Restricted Boltzmann Machine (RBM) with Practical Implementation, 2019.
- Amir Ali, Auto Encoder with Practical Implementation, 2019.

3 Zbiory danych

- MNIST Handwritten Digits
- Fashion-MNIST, Hugging Face Fashion-MNIST
- $\star$ Kuzushiji-MNIST (również Kuzushiji-49)

4 Przydatne biblioteki i funkcje

1. SciKit Learn:
    - Hugging Face: `datasets`
    - Hugging Face: `load_dataset()`

- `fetch_openml()`
- `classification_report()`
- `BernoulliRBM`, `RandomForestClassifier`
- Tensorflow: `Dense`, `Model`, `Input`

5 Ekstrakcja cech za pomocą Restricted Boltzmann Machine

Eksperymenty w tym zadaniu należy wykonać dla wszystkich zbiorów danych wymienionych w punkcie 3, przy czym obowiązkowe są zbiory MNIST i Fashion-MNIST.

- Wczytaj zbiór danych. Są to obrazy w skali szarości, przypisane do jednej z wielu klas, z pre-definiowanym podziałem na zbiór treningowy i testowy (w przypadku MNIST oraz Fashion-MNIST jest to 60000 rekordów treningowych i 10000 testowych).
- Dokonaj binaryzacji[1] obrazów (zarówno zbiór treningowy, jak i testowy), a następnie wyświel 10 przykładowych rekordów ze zbioru treningowego.
- Dokonaj konwersji danych na tablice `numpy`
- Zbuduj `Pipeline` złożony z:
  - `BernoulliRBM` do ekstrakcji cech
  - `LogisticRegression` jako klasyfikatora
- Dostrój hiperparametry modelu korzystając z wyszukiwania siatkowego i walidacji krzyżowej (`GridSearchCV`). Siatkę parametrów możesz dobrać stosowanie do zbioru danych i dostępnych zasobów obliczeniowych. Przykładowa siatka znajduje się poniżej.

```
param_grid = {
    'rbm_n_components': [70, 80, 90],  # Number of hidden units
    'rbm_learning_rate': [0.05, 0.08, 0.1],  # Learning rate
    'rbm_batch_size': [10, 20],  # Batch size
    'logistic_C': [0.1, 0.5, 1.0],  # Regularization strength
}
```

- Przedstaw i skomentuj otrzymane wartości hiperparametrów, a następnie wytrenuj zgodnie z nimi RBM i klasyfikator.
- Dokonaj klasyfikacji na zbiorze testowym i przedstaw szczegółowo wyniki (*accuracy*, *precision*, *recall*, *f1-score*):

[1] `(np.array(sample['image']) > 127).astype(np.uint8)`

  - Na poziomie każdej klasy
  - Zbiorcze

- Czy nieliniowa ekstrakcja cech za pomocą RBM poprawia wyniki klasyfikacji w porównaniu z *baseline* (regresja logistyczna na surowych pikselach)?

- Przedstaw wizualizację wszystkich ekstrachowanych cech ukrytych (`n_components` obrazów odpowiadających wyuczonym wagom łączącym się z określonym elementem warstwy ukrytej). Spróbuj rozpoznać jakie wysokopoziomowe cechy obrazu są wzmacniane przez określony komponent.

6 Hierarchiczna ekstrakcja cech za pomocą Deep Belief Network

W niniejszej sekcji sprawdzimy, czy ustawianie wielu RBM w stos, czyli budowa modelu głębokiego Deep Belief Network (i hierarchiczna ekstrakcja cech), pozwoli na poprawienie uzyskanych wcześniej wyników klasyfikacji. Podobnie jak w poprzednim zadaniu eksperymenty należy wykonać dla wszystkich zbiorów danych wymienionych w punkcie 3, przy czym obowiązkowe są zbiory MNIST i Fashion-MNIST.

1. Skonfiguruj 3 RBM w ten sposób, że warstwa ekstrakcji cech (*hidden layer*) RBM(i) staje się warstwą wejściową (*visible layer*) BRM($i + 1$). RBMy trenowane są w sekwencyjnie: 1, 2, 3.

2. Rozmiar warstwy ukrytej jest pre-definiowany, przykładowo dla MNIST proponujemy kolejno: 256, 128, 64. Dla innych zbiorów danych rozmiary te możesz dopasować empirycznie, przy czym nie jest wymagane zastosowanie wyszukiwania siatkowego (duże wymagania obliczeniowe). Wartości pozostałych hiperparametrów mogą być podobne jak w poprzednim zadaniu.

3. Porównaj wyniki klasyfikacji uzyskane z zastosowaniem regresji logistycznej, dla poniższych wariantów ekstrakcji cech:

   - *Baseline* - regresja logistyczna na surowych pikselach
   - Pełna hierarchiczna ekstrakcja (wszystkie 3 RBMy)
   - Pierwszy RBM
   - Pierwszy i drugi RBM

4. Czy ustawianie ekstraktorów RBM jest korzystne dla każdego ze zbiorów danych? W jaki przypadku hierarchiczna ekstrakcja cech może być przydatna?

7 ⋆ Restricted Boltzmann Machines: implementacja, testy rekonstrukcji

1. Zaimplementuj RBM wraz z algorytmem trenowania *Contrastive Divergence*. Możesz użyć następującego szablonu:

```
class RBM():

  def __init__(self, visible_dim, hidden_dim):
    #initialize weights
    #initialize biases

  def fit(self, X, epochs=10, batch_dim=50, lr=0.1):
    #train the Restricted Boltzmann Machine
    #interate wake and dream phases
    #wake - prob of hidden given visible
    #dream - prob of visible given hidden
    #calculate error
    #update Contrastive Divergence

  def reconstruct(self, X):
    #reconstruct the data from the hidden encoding

  def plot_weights(self, step=0):
    #visualize the learned weights for the given step
```

2. Wytrenuj RBM na zbiorze Fashion-MNIST (część treningowa) dla warstwy ukrytej (`hidden_dim`) o rozmiarze 40. Zastosuj domyślne wartości parametrów metody `fit()`.

3. Zwizualizuj jak zmieniają się wagi w kolejnych epokach treningu. W tym celi wywołuj metodę `plot_weights()` w metodzie `fit()`. Przedstaw wyniki w formie animowanego pliku GIF. Zapisz obserwacje dotyczące postępów treningu.

4. Wybierz 10 przykładów ze zbioru testowego (po jedynym dla każdej klasy). Następnie, użyj RBM do rekonstrukcji obrazu testowego. Porównaj otrzymaną rekonstrukcję z obrazem oryginalnym:

   - Przedstaw wizualizację: obraz oryginalny vs. obraz odtworzony na podstawie wyuczonych wag RBM (dla każdej klasy)
   - Ilościowo: zastosuj Structural Similarity Index (SSIM).

5. Skomentuj uzyskane wyniki.

8 Ekstrakcja cech za pomocą Autoencodera

W niniejszym zadaniu chcemy sprawdzić, czy ekstrakcja cech obrazu za pomocą różnego typu Autoencoderów może poprawić wcześniej uzyskane wyniki klasyfikacji. Będziemy rozważać zarówno proste Autoencodery, jak i modele głębokie złożone z DBN. Zadanie ma charakter otwarty - można proponować w nim własne architektury (wykorzystujące

np. architektury wariacyjne AE z poprzedniego laboratorium). Podobnie jak w poprzednim zadaniu eksperymenty należy wykonać dla wszystkich zbiorów danych wymienionych w punkcie 3, przy czym obowiązkowe są zbiory MNIST i Fashion-MNIST.

1. Wczytaj zbiór danych. Dokonaj skalowania cech do przedziału $[0, 1]$ oraz konwersji na tablice `numpy` (zarówno dane treningowe, jak i testowe).

2. Zbuduj i wytrenuj prosty Autoencoder. Przykładowa konfiguracja prostego Autoencodera:

   - Kodowanie: aktywacja ReLU
   - Dekodowanie: aktywacja sigmoidalna
   - Adam optimizer
   - Loss function: binary crossentropy
   - Encoding dimension: 196 (liczba ekstrachowanych cech)
   - Liczba epok treningu: > 30

3. Przedstaw postępy trenowania wyświetlając wartości funkcji strat (*loss*) na zbiorach treningowych i testowych dla kolejnych epok.

4. Hiperparametry Autoencodera możesz dobrać wykorzystując wyszukiwanie siatkowe (*grid search*).

5. Korzystając w wytrenowanego Autoencodera wygeneruj nowe cechy dla zbioru treningowego oraz zbioru testowego.

6. Wytrenuj klasyfikator `LogisticRegression` ustawiając `solver='newton-cg'`. Dokonaj predykcji na zbiorze testowym oraz porównaj uzyskane wyniki z wynikami z sekcji 5 i 6. Czy zmiana klasyfikatora na `RandomForestClassifier` poprawi dokładność klasyfikacji? Przedstaw swój komentarz do całościowych wyników klasyfikacji.

7. $\star$ Zaproponuj własną architekturę głębokiego Autoencodera wykorzystującego filtry konwolucyjne. Nowe podejście do ekstrakcji cech powinno poprawić dokładność klasyfikacji na wszystkich zbiorach danych.