Dokumentacja Projektu - Grafowe Sieci Neuronowe

Przemysław Dąbrowski

Przegląd Projektu

Projekt implementuje i porównuje różne architektury grafowych sieci neuronowych (GNN) na zbiorze danych Cora. Głównym celem jest analiza zachowania modeli GNN przy użyciu losowych cech węzłów zamiast oryginalnych cech, co pozwala na badanie wpływu struktury grafu na wydajność klasyfikacji.

Struktura Projektu

```
projekt/
├── Train_and_eval.py    # Główny skrypt treningowy i ewaluacyjny
├── Models.py            # Implementacje architektur GNN
├── Cora_Random_Gen.py   # Generator danych z losowymi cechami
├── Eval_util.py         # Narzędzia do szczegółowej ewaluacji
├── Cora_info.py         # Definicje klas dla zbioru Cora
└── data/                # Katalog z danymi
    ├── Cora/            # Oryginalny zbiór Cora
    └── Cora_Random/     # Zbiory z losowymi cechami
```

Szczegółowa Dokumentacja Modułów

1. Models.py - Architektury Sieci

Klasa GCN (Graph Convolutional Network)

```
class GCN(torch.nn.Module):
   def __init__(self, in_channels, hidden_channels, out_channels)
```

Parametry:

- in_channels: Liczba cech wejściowych węzłów
- hidden_channels: Rozmiar warstwy ukrytej
- out_channels: Liczba klas wyjściowych

Architektura:

- 2 warstwy konwolucyjne GCN
- Funkcja aktywacji: ReLU między warstwami
- Dropout podczas treningu
- Wyjście: log_softmax dla klasyfikacji

Zastosowanie: Standardowa architektura GCN oparta na konwolucji spektralnej, efektywna dla zadań klasyfikacji węzłów.

Klasa GAT (Graph Attention Network)

```
class GAT(torch.nn.Module):
   def __init__(self, in_channels, hidden_channels, out_channels, heads=8)
```

Parametry:

- in_channels: Liczba cech wejściowych
- hidden_channels: Rozmiar warstwy ukrytej
- out_channels: Liczba klas wyjściowych
- heads: Liczba głów mechanizmu uwagi (domyślnie 8)

Architektura:

- Warstwa 1: Multi-head attention z 8 głowami
- Warstwa 2: Single-head attention z uśrednianiem
- Funkcja aktywacji: ELU
- Dropout: 0.6 w każdej warstwie
- Wyjście: log_softmax

Zastosowanie: Wykorzystuje mechanizm uwagi do adaptacyjnego ważenia sąsiadów, szczególnie skuteczny w grafach o zróżnicowanej strukturze.

Klasa GraphSAGE

```
class GraphSAGE(torch.nn.Module):
   def __init__(self, in_channels, hidden_channels, out_channels)
```

Parametry:

- in_channels: Liczba cech wejściowych
- hidden_channels: Rozmiar warstwy ukrytej
- out_channels: Liczba klas wyjściowych

Architektura:

- 2 warstwy SAGEConv
- Funkcja aktywacji: ReLU między warstwami
- Brak log_softmax na wyjściu (surowe logity)

Zastosowanie: Induktywne uczenie - może generalizować na niewidziane węzły poprzez próbkowanie i agregację sąsiadów.

2. Cora_Random_Gen.py - Generator Danych

Funkcja generate_random_features_graph()

def generate_random_features_graph(feature_dim=None, seed=42)

Parametry:

- feature_dim: Wymiar nowych losowych cech (domyślnie jak oryginał)
- seed: Ziarno losowości dla reprodukowalności

Funkcjonalność:

1. Ładuje oryginalny zbiór danych Cora
2. Zastępuje cechy węzłów losowymi wartościami z rozkładu normalnego N(0,1)
3. Zachowuje strukturę grafu (krawędzie) i etykiety
4. Zachowuje podział train/val/test
5. Zapisuje zmodyfikowany graf do pliku pickle

Zwraca:

- modified_data: Obiekt Data z losowymi cechami
- save_path: Ścieżka do zapisanego pliku

Funkcja load_random_features_graph()

def load_random_features_graph(save_path)

Parametry:

- save_path: Ścieżka do zapisanego pliku z grafem

Funkcjonalność: Ładuje wcześniej wygenerowany graf z losowymi cechami wraz z metadanymi.

Zwraca:

- data: Obiekt Data z grafem
- metadata: Słownik z informacjami o grafie

3. Eval_util.py - Narzędzia Ewaluacji

Funkcja analyze_class_distribution()

def analyze_class_distribution(data)

Parametry:

- data: Obiekt Data z grafem

Funkcjonalność:

- Analizuje rozkład klas w zbiorach train/val/test
- Tworzy szczegółową tabelę z liczebnościami i procentami
- Wyświetla statystyki dla każdej klasy

Zwraca:

- DataFrame z analizą rozkładu klas

Funkcja calculate_random_baseline()

def calculate_random_baseline(test_labels, train_labels)

Parametry:

- test_labels: Etykiety zbioru testowego
- train_labels: Etykiety zbioru treningowego

Funkcjonalność: Oblicza bazowe dokładności dla losowego zgadywania:

- Równomierne losowanie (1/liczba_klas)
- Stratyfikowane losowanie (według rozkładu treningowego)

Zwraca:

- uniform_random_acc: Dokładność równomiernego losowania
- stratified_random_acc: Dokładność stratyfikowanego losowania

Funkcja detailed_performance_analysis()

def detailed_performance_analysis(data, pred, true_labels, mask, split_name)

Parametry:

- data: Obiekt Data z grafem
- pred: Predykcje modelu
- true_labels: Prawdziwe etykiety
- mask: Maska dla danego podziału
- split_name: Nazwa podziału ("train"/"val"/"test")

Funkcjonalność:

- Oblicza szczegółowe metryki dla każdej klasy
- Porównuje z bazowymi dokładnościami losowymi
- Wyświetla tabelę z dokładnością, precyzją, recall i F1-score
- Oblicza statystyki podsumowujące

Zwraca:

- Lista słowników z metrykami dla każdej klasy

Funkcja plot_confusion_matrix()

def plot_confusion_matrix(pred, true_labels, mask, split_name)

Parametry:

- pred: Predykcje modelu
- true_labels: Prawdziwe etykiety
- mask: Maska dla podziału
- split_name: Nazwa podziału

Funkcjonalność: Tworzy i wyświetla macierz pomyłek jako heatmapę z:

- Liczbami wystąpień dla każdej pary klas
- Nazwami klas na osiach
- Kolorową skalą reprezentującą intensywność

Funkcja plot_class_performance()

def plot_class_performance(class_metrics, split_name)

Parametry:

- class_metrics: Lista metryk dla każdej klasy
- split_name: Nazwa podziału

Funkcjonalność: Tworzy wykresy słupkowe dla:

- Dokładności per klasa
- Precyzji per klasa
- Recall per klasa
- F1-score per klasa

Funkcja analyze_predictions_confidence()

def analyze_predictions_confidence(out, mask, split_name)

Parametry:

- out: Surowe wyjście modelu (logity)
- mask: Maska dla podziału
- split_name: Nazwa podziału

Funkcjonalność:

- Analizuje pewność predykcji modelu
- Oblicza statystyki rozkładu pewności
- Kategoryzuje predykcje według poziomów pewności
- Wyświetla szczegółowe statystyki rozkładu

4. Train_and_eval.py - Główny Skrypt

Konfiguracja Eksperymentu

```
# Ładowanie danych
with open('data/Cora_Random/cora_random_features_dim1433_seed42.pkl', 'rb') as f:
    saved_data = pickle.load(f)
```

Skrypt ładuje graf z losowymi cechami zamiast oryginalnego zbioru Cora.

Wybór Modelu

```
# Dostępne konfiguracje:
# GCN: 16 ukrytych jednostek, lr=0.01, weight_decay=5e-4
# GAT: 8 ukrytych jednostek, lr=0.005, weight_decay=5e-4
# GraphSAGE: 64 ukryte jednostki, lr=0.01, weight_decay=5e-4
```

Funkcja train()

```
def train():
    model.train()
    optimizer.zero_grad()
    out = model(data.x, data.edge_index)
    loss = F.nll_loss(out[data.train_mask], data.y[data.train_mask])
    loss.backward()
    optimizer.step()
    return loss
```

Funkcjonalność:

- Ustawia model w tryb treningowy
- Wykonuje forward pass
- Oblicza negative log-likelihood loss
- Wykonuje backward pass i aktualizację wag

Funkcja test()

```
def test():
    model.eval()
    out = model(data.x, data.edge_index)
    pred = out.argmax(dim=1)
    accs = []
    for mask in [data.train_mask, data.val_mask, data.test_mask]:
        correct = pred[mask] == data.y[mask]
        acc = int(correct.sum()) / int(mask.sum())
        accs.append(acc)
    return accs
```

Funkcjonalność:

- Ustawia model w tryb ewaluacji
- Oblicza dokładności dla wszystkich podziałów
- Zwraca listę dokładności [train, val, test]

Funkcja test_detailed()

```
def test_detailed():
    # Podobna do test() ale zwraca również predykcje i surowe wyjście
    return accs, pred, out
```

Procedura Eksperymentalna

1. Przygotowanie Danych

1. Załadowanie oryginalnego zbioru Cora
2. Wygenerowanie losowych cech z rozkładu N(0,1)
3. Zachowanie struktury grafu i etykiet
4. Zapisanie zmodyfikowanego zbioru

2. Trening Modelu

1. Inicjalizacja wybranej architektury GNN
2. Konfiguracja optymalizatora Adam
3. Trening przez 200 epok
4. Monitoring dokładności co 20 epok
5. Użycie NLL loss dla GCN/GAT

3. Ewaluacja

1. **Analiza rozkładu klas**: Sprawdzenie balansu danych
2. **Szczegółowa analiza wydajności**: Metryki per klasa
3. **Analiza bazowych linii**: Porównanie z losowym zgadywaniem
4. **Analiza pewności**: Rozkład pewności predykcji
5. **Wizualizacje**: Macierze pomyłek i wykresy wydajności

4. Metryki Ewaluacyjne

- **Dokładność**: Procent poprawnych klasyfikacji
- **Precyzja**: TP/(TP+FP) dla każdej klasy
- **Recall**: TP/(TP+FN) dla każdej klasy
- **F1-score**: Harmoniczna średnia precyzji i recall
- **Pewność predykcji**: Analiza rozkładu max(softmax(logits))

Cel Badawczy

Projekt bada fundamentalne pytanie: **Ile informacji do klasyfikacji węzłów pochodzi ze struktury grafu, a ile z cech węzłów?**

Poprzez zastąpienie oryginalnych cech węzłów losowymi wartościami, można:

1. Wyizolować wpływ struktury grafu
2. Sprawdzić czy modele GNN mogą nadal skutecznie klasyfikować
3. Porównać różne architektury w kontekście wykorzystania informacji strukturalnej
4. Zrozumieć ograniczenia i możliwości modeli GNN

Wymagania Techniczne

Biblioteki

- torch - PyTorch dla deep learning
- torch_geometric - Grafowe sieci neuronowe
- sklearn - Metryki ewaluacyjne
- matplotlib, seaborn - Wizualizacje
- pandas - Analiza danych
- numpy - Operacje numeryczne

Struktura Danych

- **Graf Cora**: 2708 węzłów, 10556 krawędzi, 7 klas
- **Cechy**: 1433-wymiarowe losowe wektory z N(0,1)
- **Podział**: Standardowy train/val/test jak w oryginalnym Cora
- **Format**: PyTorch Geometric Data object