Graph Neural Networks

ANALYSE DER EINFLÜSSE VON GCN-LAYERS AUF HIDDEN REPRESENTATIONS VON NUMERISCHEN TABELLENSPALTEN

Gliederung

- Zielsetzung des Projekts
- Begriffsdefinitionen
- Vorgehen zur Lösungserarbeitung
- Endresultate
- Ausblick

Zielsetzung des Projekts

WAS WOLLEN WIR ERREICHEN?

Zielsetzung des Projekts

Einfluss der GCN-Layers auf Hidden Representations von numerischen Tabellenspalten analysieren

GCN-Modell vorstellen und Funktionsweise eines GCN-Layers erklären

Untersuchung der einzelnen Schichten des GCN und Darstellung der Vektorrepräsentationen der numerischen Spalten

Analyse der Veränderungen der Hidden Representations während des GCN-Prozesses

Zielsetzung des Projekts

Überprüfung der Vorhersagegenauigkeit des semantischen Typs mithilfe der Vektoren

Verwendung von Grafiken zur visuellen Vergleichbarkeit und Bewertung der Vektoren

Gewinnung wertvoller Erkenntnisse durch umfassende Analyse des GCN-Layers und der Hidden Representations von numerischen Tabellenspalten

Bewertung der Vorhersagegenauigkeit des semantischen Typs

Begriffsdefinitionen

GNN, CAGNN

Begriffsdefinitionen

GRAPH NEURAL NETWORKS (GNN)

Machine Learning Modelle

Graphähnliche Datenstrukturen

Beziehungen zwischen Knoten erfassen und verarbeiten

Muster in Daten erkennen

Begriffsdefinitionen

GRAPH NEURAL NETWORKS (GNN)

GNNs bestehen aus Schichten von Neuronen

Adjazenzmatrix & Knotenmerkmalmatrix als Eingaben

GNN übertragen Nachrichten zwischen Knoten → aktualisierte Darstellungen

Anwendung bei Objekterkennung, Molekularchemie & Empfehlungssysteme

Begriffsdefinitionen

CONTEXT-AWARE GRAPH NEURAL NETWORKS (CAGNN)

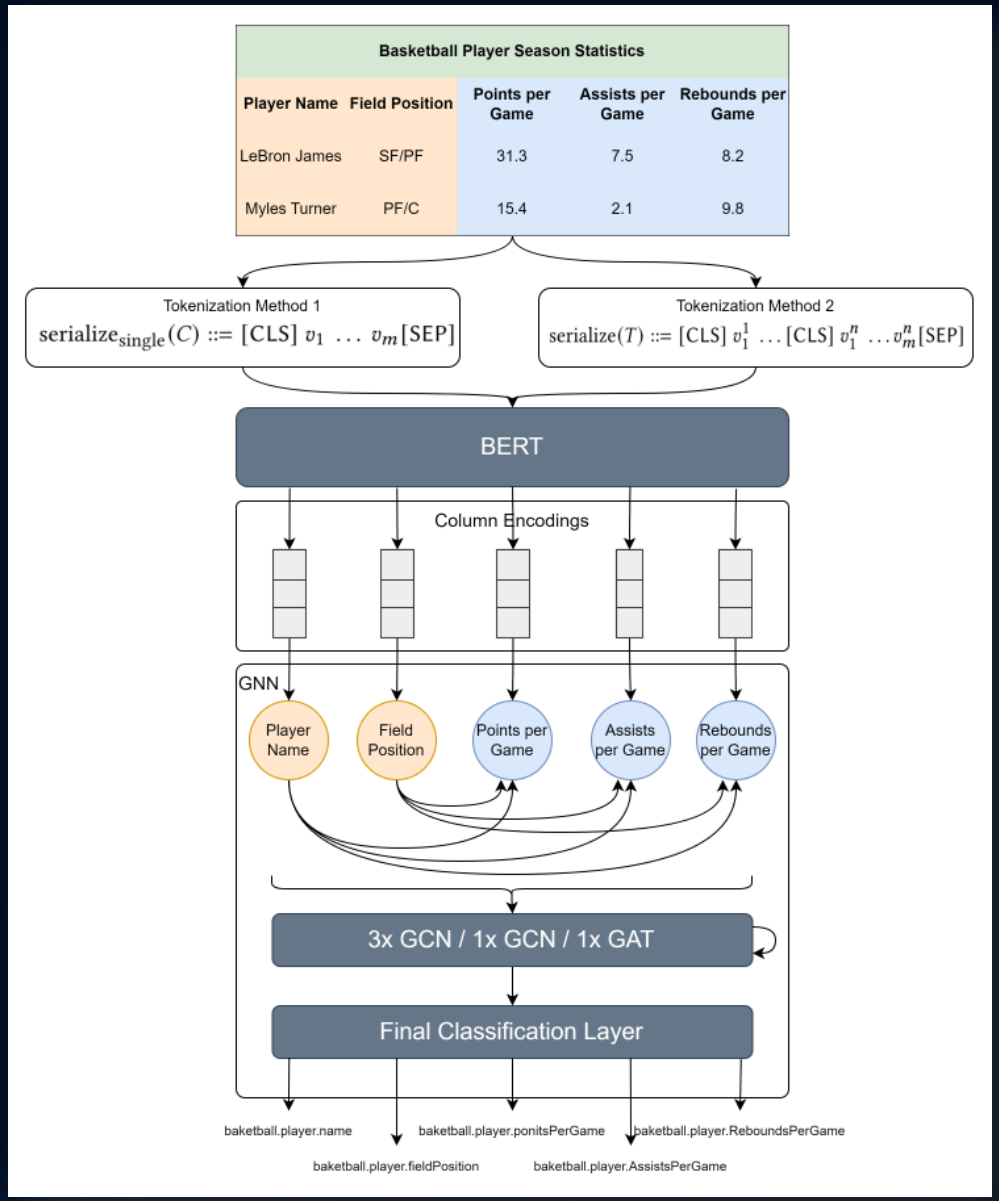
Zusätzliche Berücksichtigung des Kontexts

Verbesserte Leistung in dynamischen oder personalisierten Umgebungen

Generierung personalisierter Empfehlungen basiered auf aktuellem Kontext

Integration von Kontext erfordert Erweiterung der graphischen Darstellung

Modellvorstellung



Vorgehen zur Lösungserarbeitung

CODE & METHODEN

Vorgehen zur Lösungserarbeitung

1. Theoretisches Verständnis von GNN-Modell erhalten
2. Verständnis vom Code erhalten
   - Wie werden die Daten trainiert?
   - Wo befindet sich das Modell im Code?
   - Wie viele Schichten hat das Modell?
3. Probleme der Ausführung mussten geklärt werden

Vorgehen zur Lösungserarbeitung

4. Extrahieren der Vektoren

```
tensor([[-0.3809, -0.5406, -1.0668,  ..., -0.8276, -0.0000,  0.4662],
        [-0.3700, -0.3558, -0.8646,  ..., -0.6350, -0.3930,  0.3247],
        [-0.6058, -0.3536, -0.7674,  ..., -0.4722, -0.5063,  0.6135],
        [-0.0000, -0.6006, -0.9716,  ..., -0.8779, -0.7272,  0.8882],
        [-0.4106, -0.6216, -1.0919,  ..., -1.0296, -0.5274,  0.4273]],
       grad_fn=<MulBackward0>)
```

```python
def forward(self, graph): # hier anpassen weitere parameter table_names, column_names
    # foward pass of the bert model
    input_ids = graph.ndata["data_tensor"]
    output = self.bert(input_ids=input_ids)
    pooled_output = output[1]
    pooled_output = self.dropout(pooled_output)

    # extract pooled output
    input_layer = []
    for element in pooled_output:
        input_layer.append(element.tolist())

    # hidden_state_output = self.dropout(output["last_hidden_state"])
    # maybe use pooled output from bert?
    # polled_output = self.dropout(output[1]) # pooler output is at position 1
    # GNN forward pass
    h_one = self.conv1(graph, pooled_output) # hidden state of CLS token is on first position
    h_one = F.relu(h_one)
    # extract h_one
    hidden_layer1 = []
    for element in h_one:
        hidden_layer1.append(element.tolist())

    h_two = self.conv2(graph, h_one)
    h_two = F.relu(h_two)
    # extract h_two
    hidden_layer2 = []
    for element in h_two:
        hidden_layer2.append(element.tolist())

    gnn_output = self.conv3(graph, h_two)
    # extract gnn_output
    output_layer = []
    for element in gnn_output:
        output_layer.append(element.tolist())
```

Vorgehen zur Lösungserarbeitung

5. Speichern der Vektoren in eine CSV-Datei

```
self.csv_file = '/ext/daten-wi/wi20b/column_annotation_gnn/extracted_vector/extracted_vectors.csv'
self.file_exists = False
with open(self.csv_file, 'a', newline='') as csvfile:
    # Check if the file exists and determine whether to write the header
    self.file_exists = csvfile.tell() != 0
    self.writer = csv.writer(csvfile)
    if not self.file_exists:
        header = ["input_layer", "hidden_layer1", "hidden_layer2", "output_layer"]
        self.writer.writerow(header)
```

```
table_size = len(pooled_output)
table_counter = 0
# save vectors in a CSV file
with open(self.csv_file, 'a', newline='') as csvfile:
    writer = csv.writer(csvfile)

    for element in range(table_size):
        extracted_vectors = []
        extracted_vectors.append(input_layer[table_counter])
        extracted_vectors.append(hidden_layer1[table_counter])
        extracted_vectors.append(hidden_layer2[table_counter])
        extracted_vectors.append(output_layer[table_counter])
        writer.writerow(extracted_vectors)
        table_counter += 1

return gnn_output
```

Vorgehen zur Lösungserarbeitung

6. Tabellennamen, Spaltennamen und Datentyp in die CSV

```
with open("/ext/daten-wi/wi20b/column_annotation_gnn/extracted_vector/output_tabledata.csv", 'a', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for i in range(10):
        writer.writerows(self.df[['table_name', 'column_name', 'columns_data_type']].values.tolist())
```

7. Zusammenfügen mit der Vektoren CSV-Datei

```
# Open the first CSV file for reading
headers = ["table_name", "column_name", "columns_data_type"]
with open('/ext/daten-wi/wi20b/column_annotation_gnn/extracted_vector/output_tabledata.csv', 'r') as file1:
    csv_reader1 = csv.reader(file1)
    data1 = list(csv_reader1)
    data1.insert(0, headers)

# Open the second CSV file for reading.
with open('/ext/daten-wi/wi20b/column_annotation_gnn/extracted_vector/extracted_vectors.csv', 'r') as file2:
    csv_reader2 = csv.reader(file2)
    data2 = list(csv_reader2)

# Merging the data
merged_data = [row1 + row2 for row1, row2 in zip(data1, data2)]

# Open the analyse file for writing.
with open('/ext/daten-wi/wi20b/column_annotation_gnn/extracted_vector/analyse.csv', 'w', newline='') as target_file:
    csv_writer = csv.writer(target_file)

    # Write the merged data to the target file.
    for row in merged_data:
        csv_writer.writerow(row)
```

Vorgehen zur Lösungserarbeitung

7. Zusammengefügte CSV-Datei

	table_name	column_name	columns_data_type	input_layer	hidden_layer1	hidden_layer2	output_layer
0	mlb_season_standings_2002	Tm	textual	[-0.5410828590393066, -0.39178478717803955, -0...	[0.0, 0.2690604329109192, 0.0, 0.2246376872062...	[0.06139879301190376, 0.035625435411930084, 0....	[0.030486760661005974, -0.04748900607228279, 0...
1	mlb_season_standings_2002	W	numerical	[-0.5187528729438782, -0.26157864928245544, -0...	[0.0, 0.6606199145317078, 0.12713615596294403,...	[0.11953523755073547, 0.15661540627479553, 0.0...	[0.24039588868618011, -0.3075655996799469, 0.4...
2	mlb_season_standings_2002	L	numerical	[-0.5030636787414551, -0.27531126141548157, -0...	[0.0, 0.6458011865615845, 0.23058900237083435,...	[0.240712508559227, 0.1091676875948906, 0.0, 0...	[0.1743149757385254, -0.3349403440952301, 0.43...
3	mlb_season_standings_2002	W-L%	numerical	[-0.7395317554473877, -0.5471993684768677, -1....	[0.0, 0.7859812378883362, 0.0, 0.9968171119689...	[0.2719872295856476, 0.3850599527359009, 0.0, ...	[0.20126932859420776, -0.37531712651252747, 0....
4	mlb_season_standings_2002	GB	numerical	[-0.5902543663978577, -0.6164579391479492, -1....	[0.0, 0.26510265469551086, 0.0, 0.967561662197...	[0.40234294533729553, 0.44962307810783386, 0.0...	[0.1456291675567627, -0.3064262866973877, 0.48...
5	mlb_season_standings_2002	Tm	textual	[-0.5939173102378845, -0.4165181517601013, -0....	[0.0, 0.05148514360189438, 0.0, 0.060327041894...	[0.009123655967414379, 0.004252934362739325, 0...	[0.0014558505499735475, -0.00253783049993217, ...

Vorgehen zur Lösungserarbeitung

Analysemethode

- T-SNE
  - Kommt immer wieder zu Problemen, eine Analyse ist nicht möglich
  - Ständige neue Fehler haben die Arbeit mit dieser Analysemethode unmöglich gemacht
  - Mögliche Erklärung: Vektoren sind nicht 2 Dimensional

```python
# Pfad zur CSV-Datei
csv_path = "analyse.csv"

# DataFrame aus der CSV-Datei laden
df = pd.read_csv(csv_path, nrows=50)
df = df.drop(columns=["table_name", "column_name", "columns_data_type"])

# Umwandeln der Listen in den Spalten in numerische Werte
for col in df.columns:
    if df[col].dtype == object:
        df[col] = df[col].apply(lambda x: np.mean(eval(x)))

# Vektoren aus dem DataFrame laden und konvertieren
vectors = df.values.astype(np.float64)

# t-SNE auf die Vektoren anwenden
perplexity = min(30, vectors.shape[0] - 1)  # Beispielwert für die Perplexität
tsne = TSNE(n_components=2, random_state=42, perplexity=perplexity)
vectors_tsne = tsne.fit_transform(vectors)

# Scatterplot der t-SNE-Darstellung der Vektoren
plt.scatter(vectors_tsne[:, 0], vectors_tsne[:, 1])
plt.show()
```

Vorgehen zur Lösungserarbeitung

Analysemethode

- PCA (Principal Component Analysis)
  - Reduziert die Dimensionalität (Vektoren) auf 2 Hauptkomponenten

```python
# Daten aus der CSV-Datei laden
df = pd.read_csv(csv_path, nrows=1000)
# Die Namen der Spalten
col_names = df.columns[3:]
# Funktion zum Umwandeln der Listen in numerische Werte
def parse_list(x):
    try:
        return ast.literal_eval(x)
    except (ValueError, SyntaxError):
        return []
# Eine Liste von Farben für die verschiedenen Scatterplots
colors = ['red', 'green', 'blue', 'purple']
# Anzahl der Plots pro Zeile und Spalte
num_cols = 2
num_rows = int(np.ceil(len(col_names) / num_cols))
# Figur und Achsen für die Scatterplots erstellen
fig, axs = plt.subplots(num_rows, num_cols, figsize=(12, 8))
# Für jeden Layer im DataFrame
for i, col in enumerate(col_names):
    # Daten für den aktuellen Layer extrahieren und in den richtigen Datentyp konvertieren
    data = df[col].apply(parse_list).values
    vectors = np.array([np.array(x) for x in data])
    # PCA-Objekt erstellen
    pca = PCA(n_components=2)
    # PCA an die Daten anpassen und sie transformieren
    transformed_data = pca.fit_transform(vectors)
    # Aktuelle Achse auswählen
    ax = axs[i // num_cols, i % num_cols]
    # Scatterplot erstellen
    ax.scatter(transformed_data[:, 0], transformed_data[:, 1], color=colors[i])
    # Titel hinzufügen
    ax.set_title(col)
# Die Plots anzeigen
plt.show()
```

Vorgehen zur Lösungserarbeitung

Analysemethode

- PCA (Principal Component Analysis) für alle Vektoren einer Zeile
  - Innerhalb einer Zeile eines Layers befinden sich mehrere Hundert Vektoren
  - Hier wird jeder einzelne Vektor durch die PCA berücksichtig
  - Da die Vektoren allerdings 1-Dimensional sind, stehen die Punkte im Plot nur auf einer Linie und nicht im 2 Dimensionalen Raum

```
df = pd.read_csv(csv_path, nrows=1)
col_names = df.columns[3:]

def parse_list(x):
    try:
        return ast.literal_eval(x)
    except (ValueError, SyntaxError):
        return []

colors = ['red', 'green', 'blue', 'purple']
num_cols = 2
num_rows = int(np.ceil(len(col_names) / num_cols))
fig, axs = plt.subplots(num_rows, num_cols, figsize=(8, 8))

for i, col in enumerate(col_names):
    data = df[col].apply(parse_list).values.tolist()

    vectors = np.array([item for sublist in data for item in sublist]).reshape(-1, 1)

    pca = PCA(n_components=1)
    transformed_data = pca.fit_transform(vectors)
    ax = axs[i // num_cols, i % num_cols]

    ax.scatter(transformed_data, [0]*len(transformed_data), color=colors[i])
    ax.set_title(col)
    ax.set_xlabel('Principal Component 1')

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

Vorgehen zur Lösungserarbeitung

Analysemethode

- PCA (Principal Component Analysis)
  - Zum besseren Vergleich werden hier alle Layer in einem Plot ausgegeben

```python
# Daten aus der CSV-Datei laden
df = pd.read_csv(csv_path, nrows=1000)
# Die Namen der Spalten
col_names = df.columns[3:]
# Funktion zum Umwandeln der Listen in numerische Werte
def parse_list(x):
    try:
        return ast.literal_eval(x)
    except (ValueError, SyntaxError):
        return []
# Eine Liste von Farben für die verschiedenen Scatterplots
colors = ['red', 'green', 'blue', 'purple']
# Figur und Achse für den kombinierten Plot erstellen
fig, ax = plt.subplots(figsize=(8, 6))
# Für jeden Layer im DataFrame
for i, col in enumerate(col_names):
    # Daten für den aktuellen Layer extrahieren und in den richtigen Datentyp konvertieren
    data = df[col].apply(parse_list).values
    vectors = np.array([np.array(x) for x in data])
    # PCA-Objekt erstellen
    pca = PCA(n_components=2)
    # PCA an die Daten anpassen und sie transformieren
    transformed_data = pca.fit_transform(vectors)
    # Scatterplot erstellen
    ax.scatter(transformed_data[:, 0], transformed_data[:, 1], color=colors[i], label=col)
# Legende hinzufügen
ax.legend()
# Titel setzen
ax.set_title("PCA Scatterplot für die vier Layer")
# Den Plot anzeigen
plt.show()
```

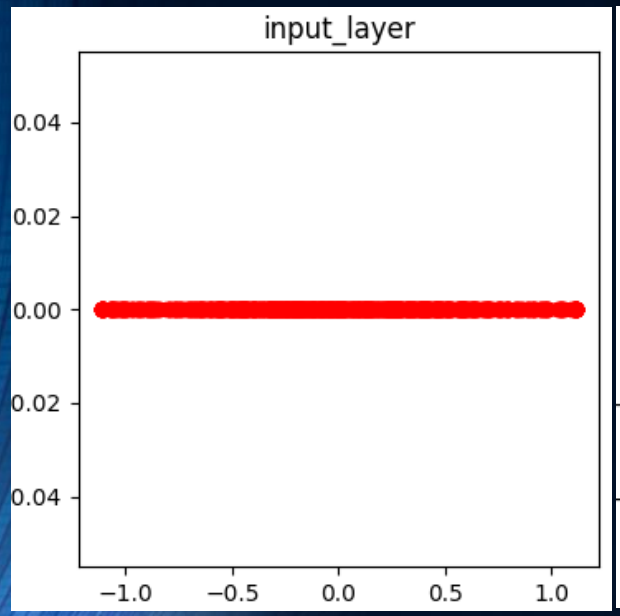
Endresultate

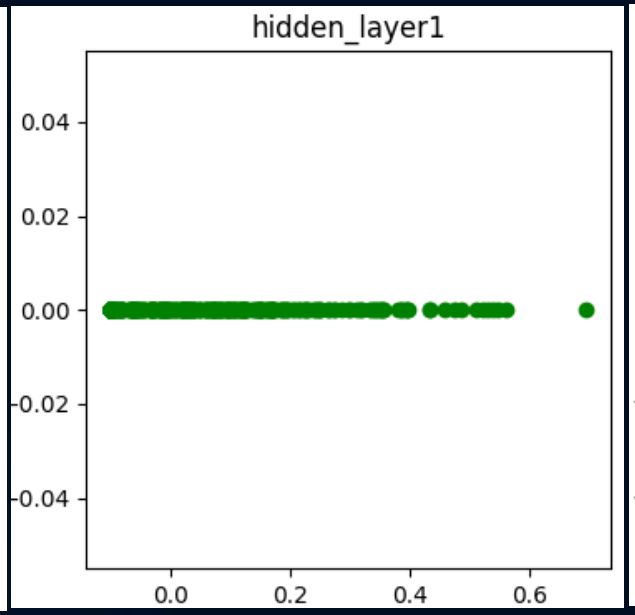
ERGEBNISSE & AUSWERTUNGEN

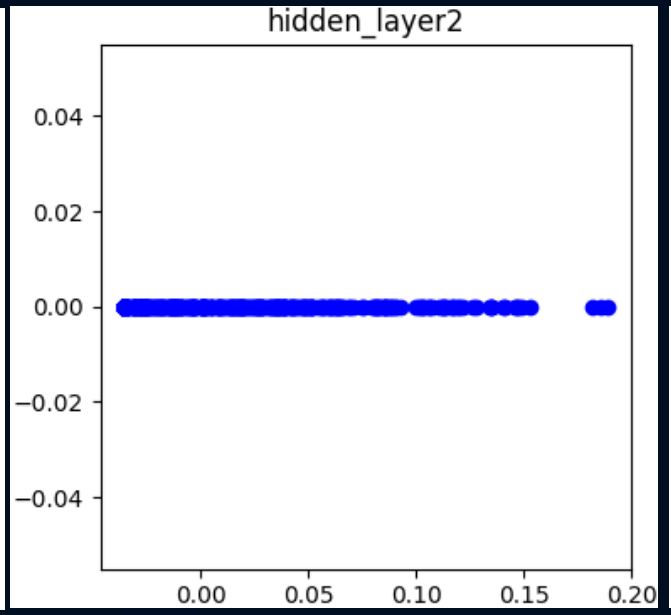
Endresultate

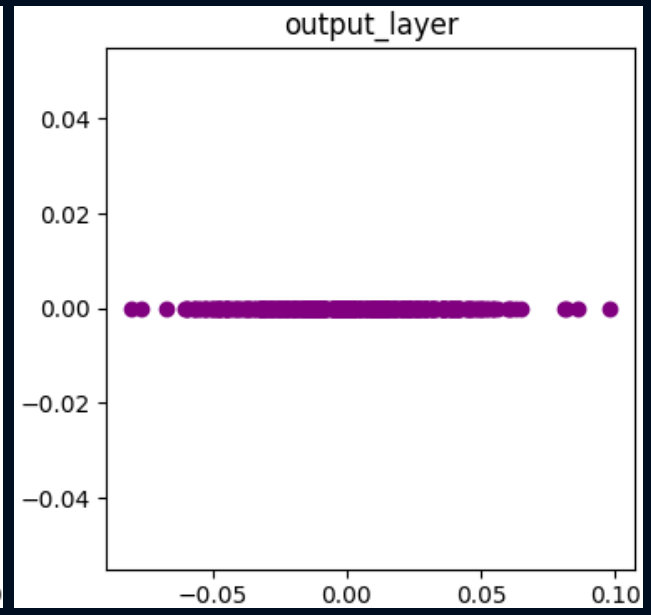
Analysemethode

- PCA (Principal Component Analysis) für alle Vektoren einer Zeile





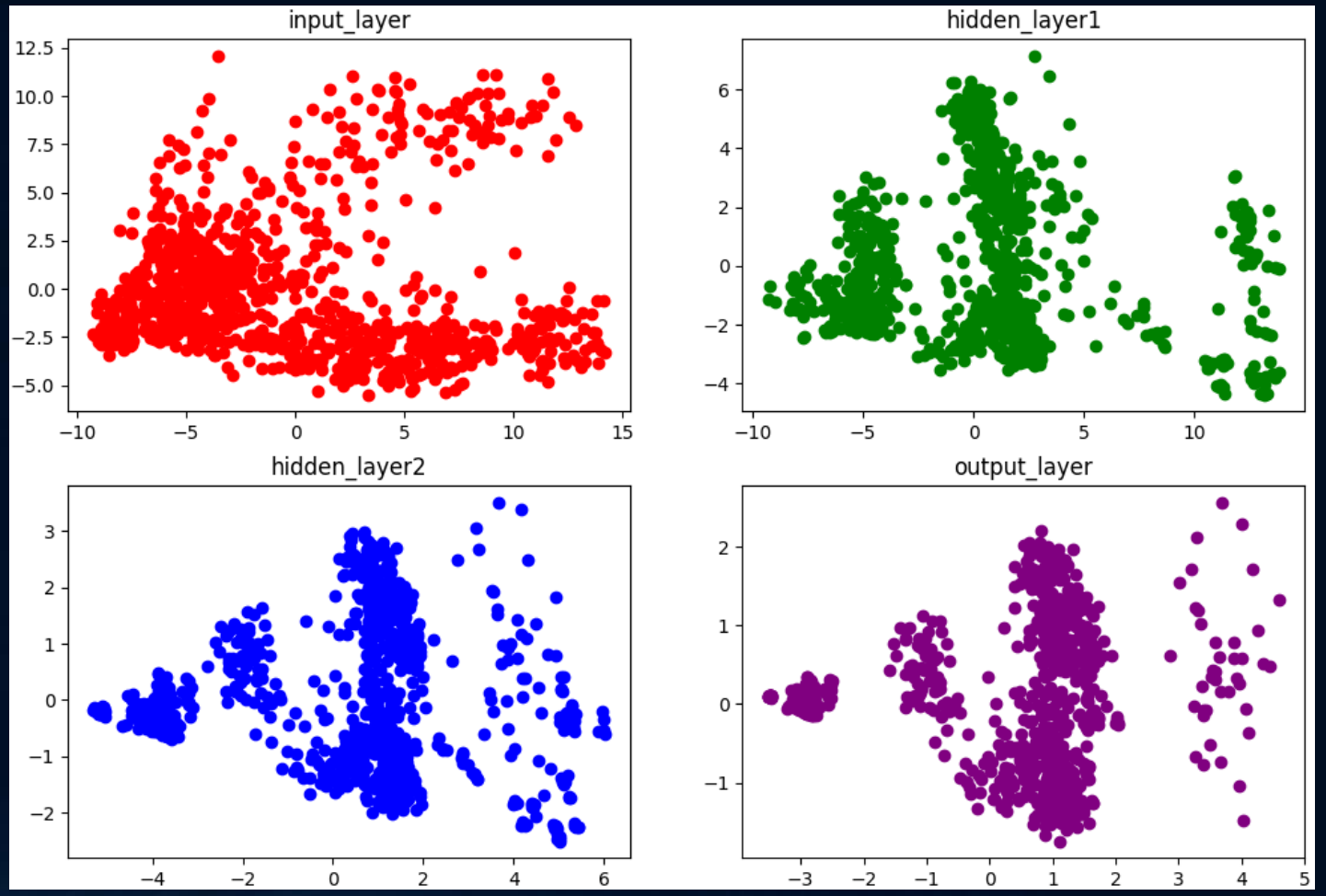




Endresultate

Analysemethode

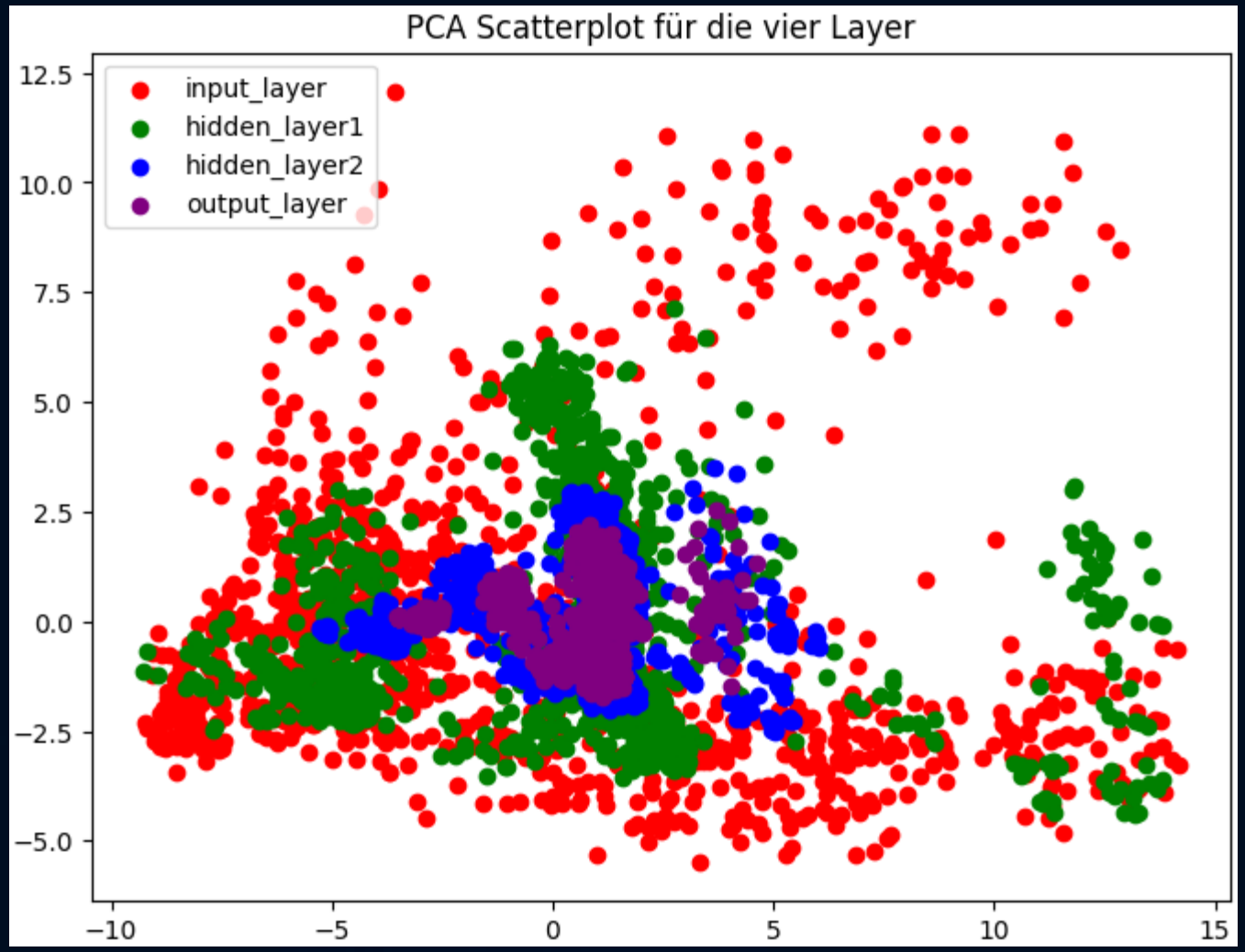
- PCA (Principal Component Analysis)



Endresultate

Analysemethode

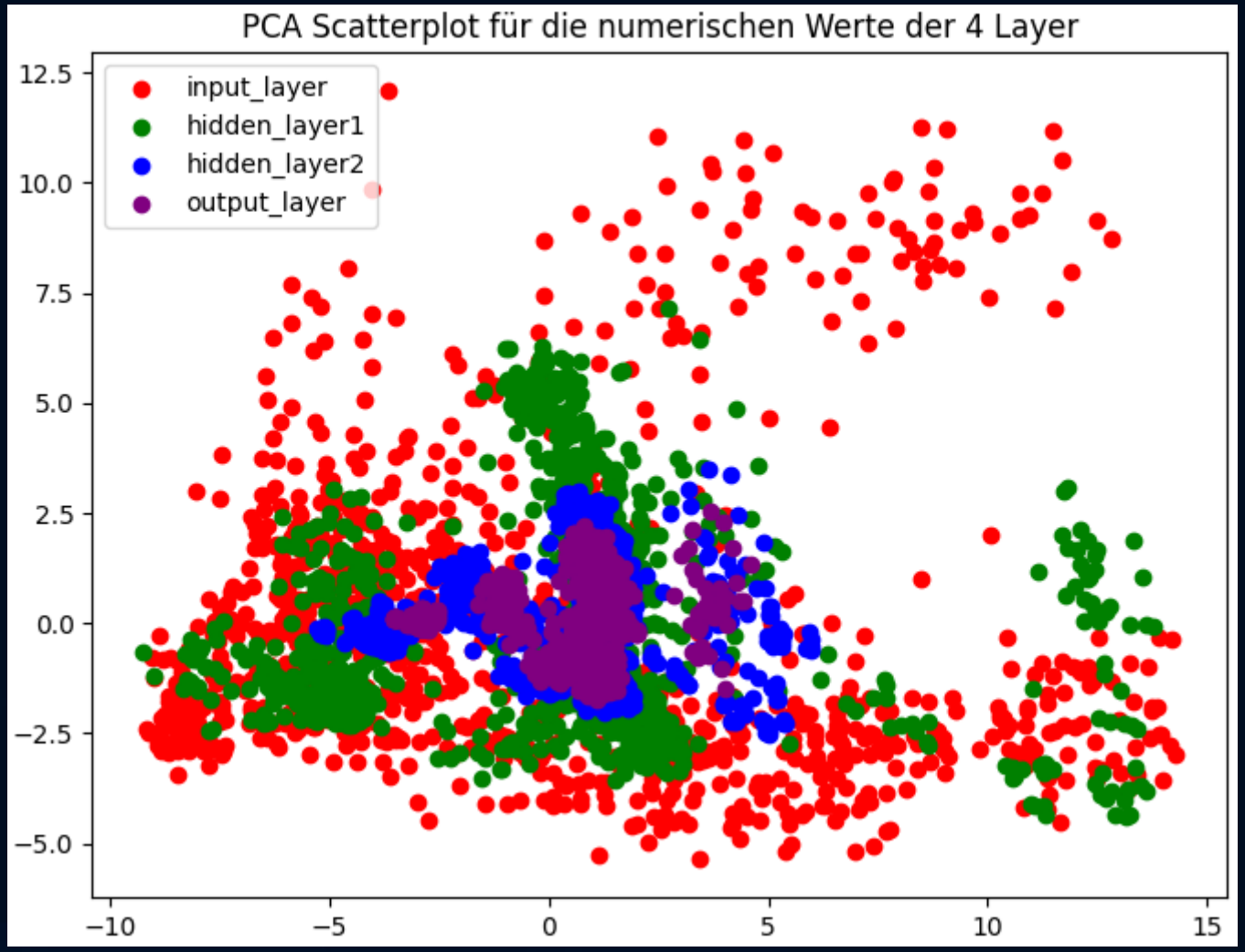
- PCA (Principal Component Analysis)
  - Im Input Layer sind die Vektoren noch im Raum verteilt
  - Beim Durchgang der Hidden Layer clustern sich die Vektoren immer weiter
  - Im Output Layer sind die Vektoren in ihren jeweiligen Cluster eng gruppiert



Endresultate

Analysemethode

- PCA (Principal Component Analysis)
  - Hier wird die Analyse nur auf die numerischen Zeilen durchgeführt
  - Nur eine geringe Änderung der Punkte



Ausblick

NUTZUNG & ANWENDUNG

Ausblick

Korrekte Vorhersage des semantischen Typs

Anwendung in verschiedenen Kontexten

Aussagen über Leistung möglich (z.B. Rangliste)

Ausblick

Datenanalyse

Entscheidungsfindung

ERLEICHTERN

Vielen Dank für Ihre Aufmerksamkeit!

DANIEL BOGER, JULIAN STIPOVIC, FABIAN QARQUR, DAMIEN ARRIENS, SIMON DI LATTE