UNIVERSITÄT
LEIPZIG

Verteidigung der Masterarbeit

Development and Evaluation of Particle Swarm Optimization Strategies for the Training of Artificial Neural Networks

Leipzig, 18.12.2025

Lukas Hein

Gliederung

1. **Ziele** der Arbeit
2. **Grundlagen** des Maschinellen Lernens
3. **PSO-Adam** und die zugrundeliegenden Optimierungsalgorithmen
4. **Implementierung** in Tensorflow mit Keras
5. **Experimente**, deren Aufbau und Auswertung
6. **Erweiterungen** des Algorithmus
7. **Diskussion**

https://github.com/Lukarion2000/PSO-Adam

1. ZIELE DER ARBEIT

1. Ziele der Arbeit

1. Entwicklung eines hybriden **Optimierungsalgorithmus** aus **Adam** und **PSO** für das **Training Neuronaler Netze**

2. Implementierung in der **Keras API** in Tensorflow

3. Untersuchung der Hyperparameter

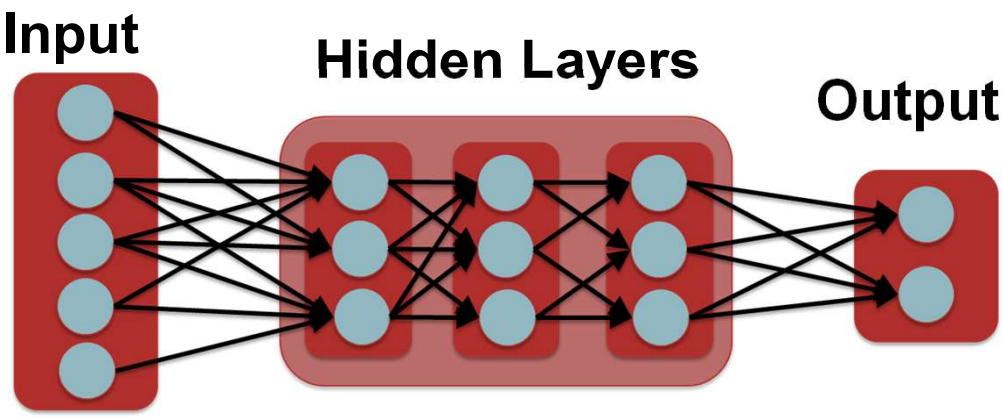
2. GRUNDLAGEN

des Maschinellen Lernens

2. Grundlagen – Künstliche Neuronale Netze

Künstliche Neuronale Netze (**KNN**) – Artificial Neural Networks (**ANN**)

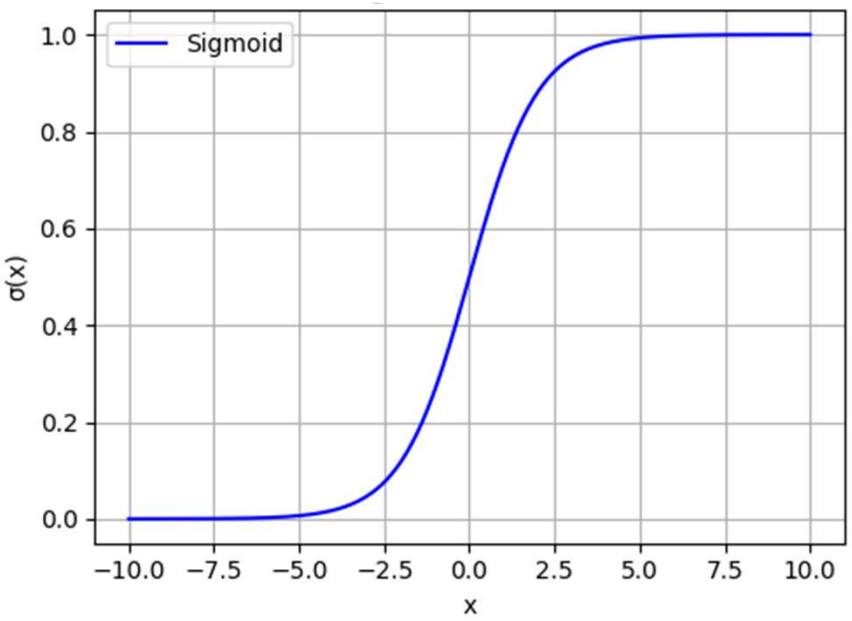
- durch Nervenzellen Inspiriert
- Darstellung komplexer Funktionen



Aufbau Feedforward Netzwerke:

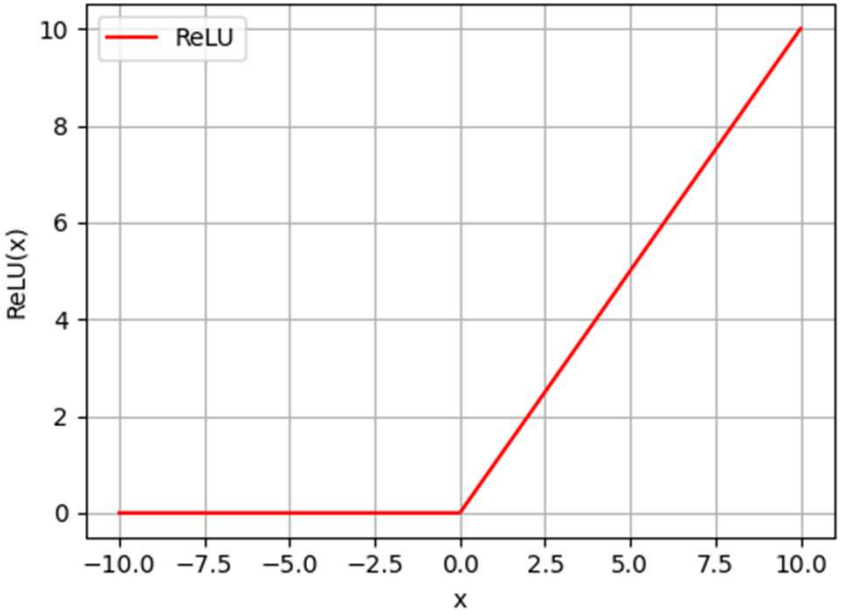
- Signale werden in eine Richtung weitergeleitet
- Schichten aus Neuronen mit
  - nichtlinearen Aktivierungsfunktionen
  - eigenem Bias
  - gewichteten Verbindungen
- $Signal_{out}$ = Aktivierungsfunktion($Signale_{in}$ * Gewichte + Bias)

2. Grundlagen – Aktivierungsfunktionen



Sigmoid

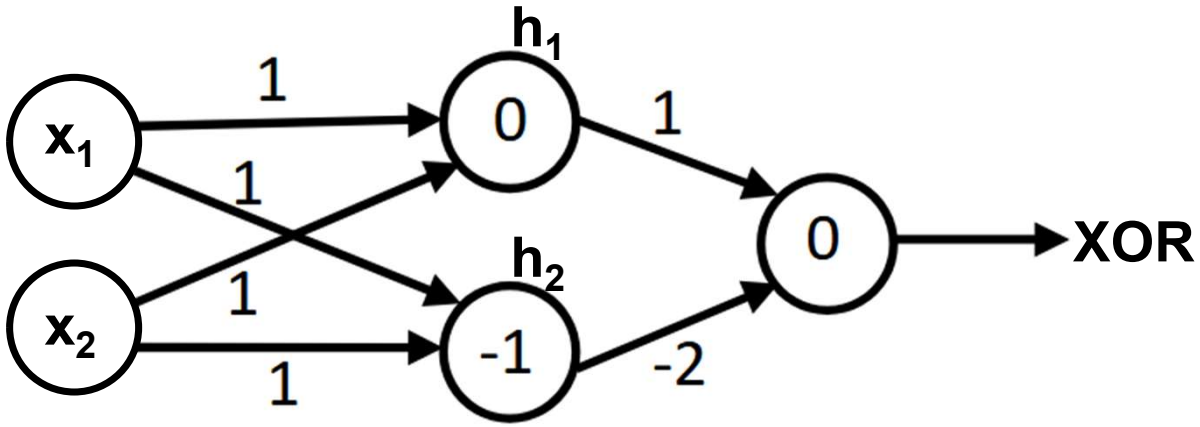
- Abbildung des Signals auf (0, 1)
- Anwendung für Klassifikation
- Softmax:
  - Spezialfall für mehrere Klassen

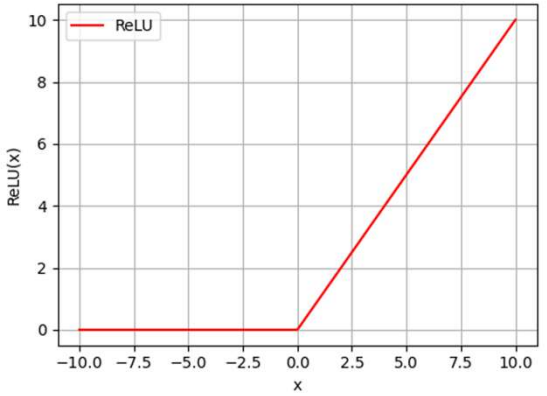


ReLU

- Negative Werte auf 0 abgebildet
- Vermindert verschwindende Gradienten
- In Hidden Layers verwendet

2. Grundlagen – KNN zum Lösen von XOR





Input		Hidden Layer		Output
x_1	x_2	h_1	h_2	XOR
0	0	**0**	**0**	**0**
0	1	**1**	**0**	**1**
1	0	**1**	**0**	**1**
1	1	**2**	**1**	**0**

2. Grundlagen – Training Künstlicher Neuronaler Netze

- ➢ Anpassung der Modellparameter durch wiederholte Evaluation von Datensätzen
  - z.B. Bilder mit Labels, Messwerte aus Experimenten
  - KNN „erlernt“ passende Funktion, die die gegebenen Datenpaare beschreibt

Trainingsdaten:
- Training des Netzes
- Aufteilung in Batches (je ein Trainingszyklus pro Batch)

Validierungsdaten:
- Überwachen des Trainingsfortschritts (Accuracy, Loss)

Testdaten:
- Überprüfen der finalen Performanz

2. Grundlagen – Lossfunktion

- KNN mit **n** Modellparametern (Verbindungsgewichte, Biases)
- Trainingsdaten ergeben eine Lossfunktion mit **n** Dimensionen
  - Differenz zwischen „wahren Outputs“ aus den Trainingsdaten und ermittelten Werten des KNN
  - Unterschiedliche Trainingsbatches ergeben unterschiedliche Losslandschaften mit Minima, Sattelpunkten, etc.

- Das Training eines KNN kann als **Optimierungsproblem** aufgefasst werden.
  - Das Globale Minimum beschreibt die besten Modellparameter.

2. Grundlagen – Trainingszyklus

➢ Je ein Trainingszyklus pro Trainingsdatenbatch

1. **Forward-Pass**
   - Erzeugen von Outputs durch Eingabe der Trainingsdaten in das Netzwerk
2. **Anwenden der Lossfunktion**
   - Fehler des ermittelten Outputs bezüglich der Trainingsdaten
   - Abhängig von Modellparametern
3. **Backward-Pass** (Backpropagation)
   - Berechnung der Lossfunktion-Gradienten bezüglich der Modellgewichte
   - Gradienten geben die Richtung des Steilsten Anstiegs in der Funktionslandschaft vor
   - Negative Gradienten beschreiben, wie Parameter verändert werden müssen
4. **Parameterupdate**
   - Optimierungsalgorithmus verändert iterativ Modellgewichte

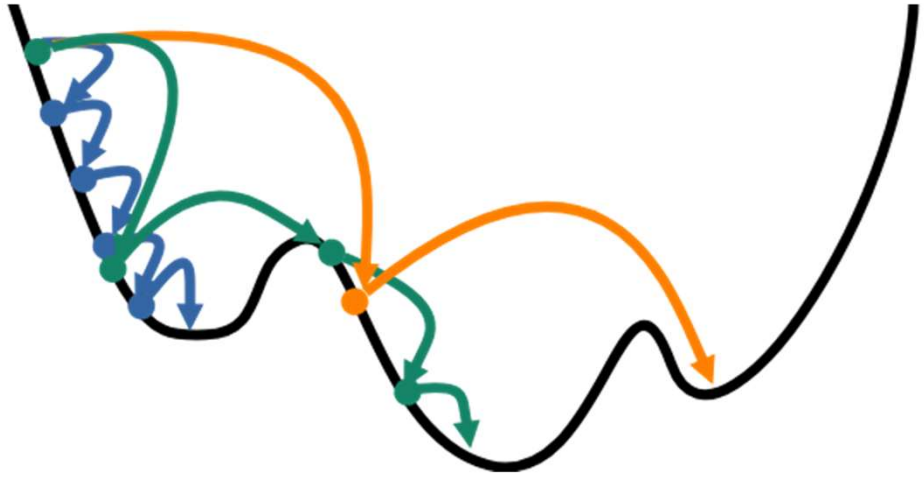
2. Grundlagen – Optimierungsalgorithmen

- Optimieren Funktionen durch Iterative Bewegung über die Funktionslandschaft
- ➢ Gradientenverfahren folgen der Richtung des negativen Gradienten.
  - Schritt wird durch Lernrate skaliert
- Konvergieren an Extremstellen (Gradient wird 0)

: zu groß

: optimal

: zu klein



3. PSO-ADAM

und die zugrundeliegenden Optimierungsalgorithmen

3. PSO-Adam – Adaptive Moment Estimation (Adam)

Kingma, D.P. & Ba, J. (2015). *Adam: A method for stochastic optimization.*

- Gradientenverfahren
- Kombination von Momentum und Adaptiven Lernraten
  - Bilden von Mittelwerten über bisherige Gradienten
  - Skalierung der Schritte unter Berücksichtigung der Gradienten-Varianz um Overshooting und Steckenbleiben zu vermeiden
- Standardoptimierer in PyTorch und TensorFlow

3. PSO-Adam – Adaptive Moment Estimation (Adam)

Gradient: (aus Backpropagation)
$$g_t = \nabla_\theta f_t(\theta_{t-1})$$

1. Moment (Momentum):
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

2. Moment(Varianz):
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Bias-Korrekturen: (Verminderung des Einflusses des Startwerts „0“)
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Parameter Update:
$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Parameter und Standardwerte:

Lernrate	α	0,001
Moment-verfallsraten	β_1	0,900
	β_2	0,999

3. PSO-Adam – Partikelschwarmoptimierung (PSO)

Kennedy, J. & Eberhart, R. (1995). *Particle swarm optimization.*

- Nutzt keine Gradienten
- Vorbild: Fisch- und Vogelschwärme auf Nahrungssuche
- Partikel / Agenten ermitteln den Funktionswert an ihrer Position
  - ➢ bewegen sich in Richtung des besten bereits gefundenen Funktionswertes innerhalb ihrer Nachbarschaft
- Mittlerweile Hauptsächlich Anwendung für Hyperparametertuning

3. PSO-Adam – Partikelschwarmoptimierung (PSO)

- Initialisierung an zufälligem Startpunkt mit zufälliger Geschwindigkeit

Geschwindigkeitsupdate:

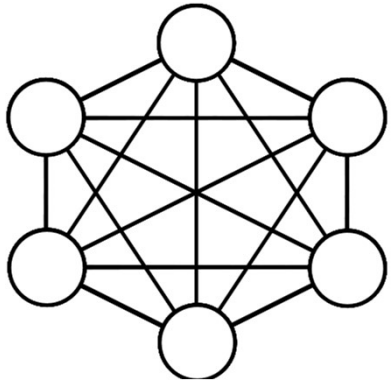
$$v_{i,t+1} = \underbrace{w \cdot v_{i,t}}_{\text{Trägheitsterm}} + \underbrace{c_1 \cdot r_1 \cdot (pbest_i - x_{i,t})}_{\text{kognitive Komponente}} + \underbrace{c_2 \cdot r_2 \cdot (gbest - x_{i,t})}_{\text{soziale Komponente}}$$

Positionsupdate: $x_{i,t+1} = x_{i,t} + v_{i,t+1}$

Parameter und gängige Werte:

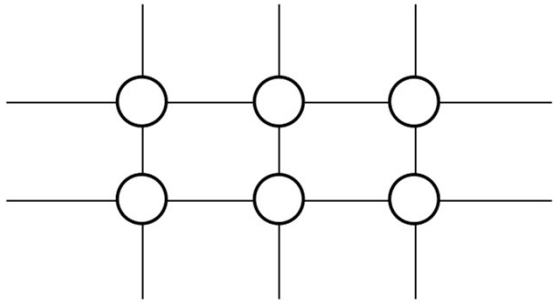
Inertia Weight	w	0,7
kognitiver Koeffizient	c_1	1,5
sozialer Koeffizient	c_2	1,5
Zufallswerte	r_1 r_2	(0, 1)

3. PSO-Adam – Beispiele für PSO Nachbarschaften



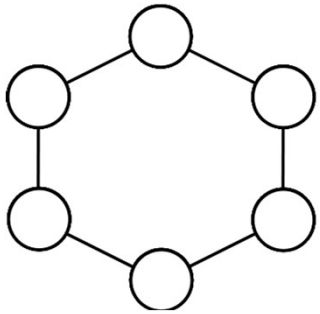
gbest

- Alle Partikel miteinander vernetzt
- Soziale Bewegung in Richtung des **Global Best**



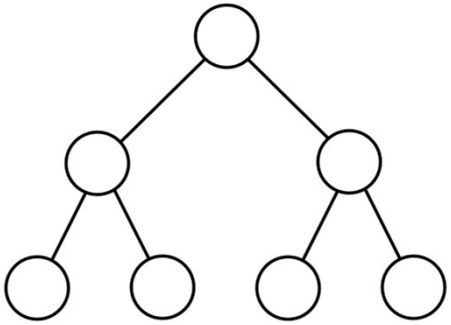
von-Neumann

- Partikel in Gitterstruktur vernetzt
- Jeder Partikel hat 4 Nachbarn



Ring / lbest

- Jedes Partikel besitz genau 2 Nachbarn
- Langsamer Austausch von Informationen



Baum

- Nutzung in Hierarchischem PSO
- Partikel steigen in der Hierarchie auf/ab und beeinflussen ihre Kinderknoten

3. PSO-Adam – Konzept des Algorithmus

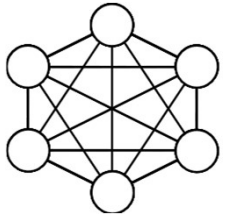
Praxis: Mehrere Trainingsläufe

- Schlecht initialisierte Durchläufe bleiben in lokalen Minima stecken

Idee: verschiedene Trainingsläufe als Partikel eines Schwarms

- Steckenbleibende Partikel werden aus ihren Positionen herausgezogen
- Explorieren unerforschte Gebiete der Funktionslandschaft

- **Mehrere Adam-Optimierer als Partikel eines Schwarms**
  - Adam-Schritt als antreibende Kraft
  - PSO-Schritt zur Unterstützung
  - Nutzung von gbest als Nachbarschaft



3. PSO-Adam – Ablauf des Algorithmus

Je Partikel $\mathbf{x_i}$:

1. **PSO-Schritt** aus **kognitiver** und **sozialen Komponente** berechnen

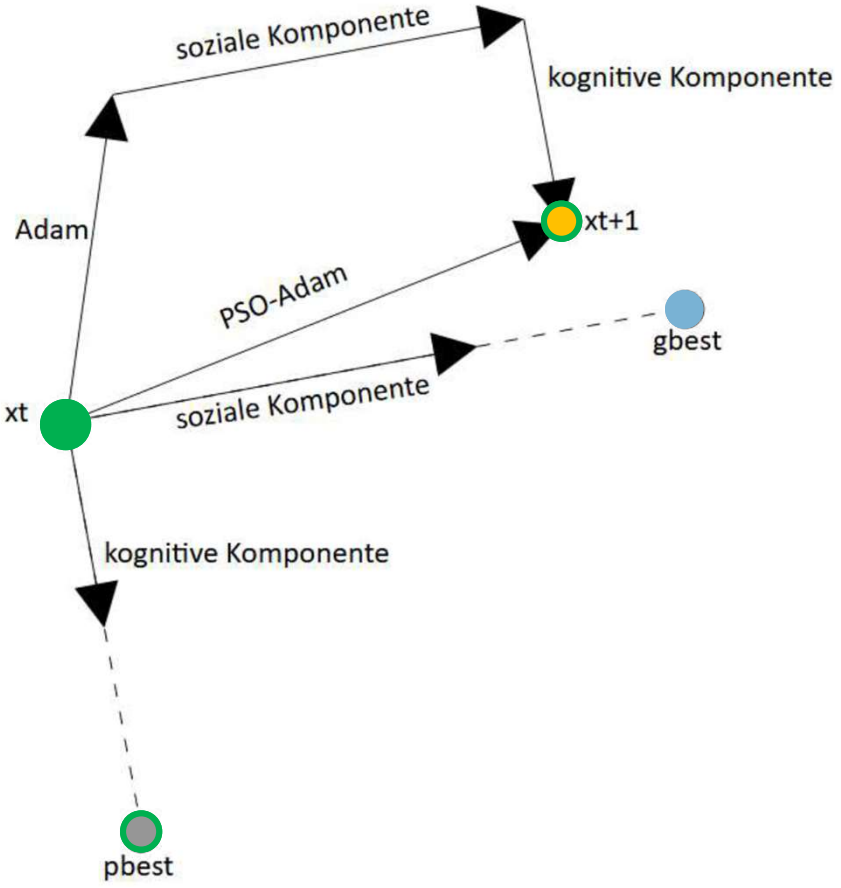
$$\Delta_{\text{PSO}} = c_1 \cdot r_1 \cdot (pbest_i - x_{i,t}) + c_2 \cdot r_2 \cdot (gbest - x_{i,t})$$

2. **Adam-Schritt** berechnen

$$\Delta_{\text{Adam}} = -\alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

3. **PSO-** und **Adam-Schritt** auf Position addieren

$$x_{i,t+1} = x_{i,t} + \Delta_{\text{Adam}} + \Delta_{\text{PSO}}$$



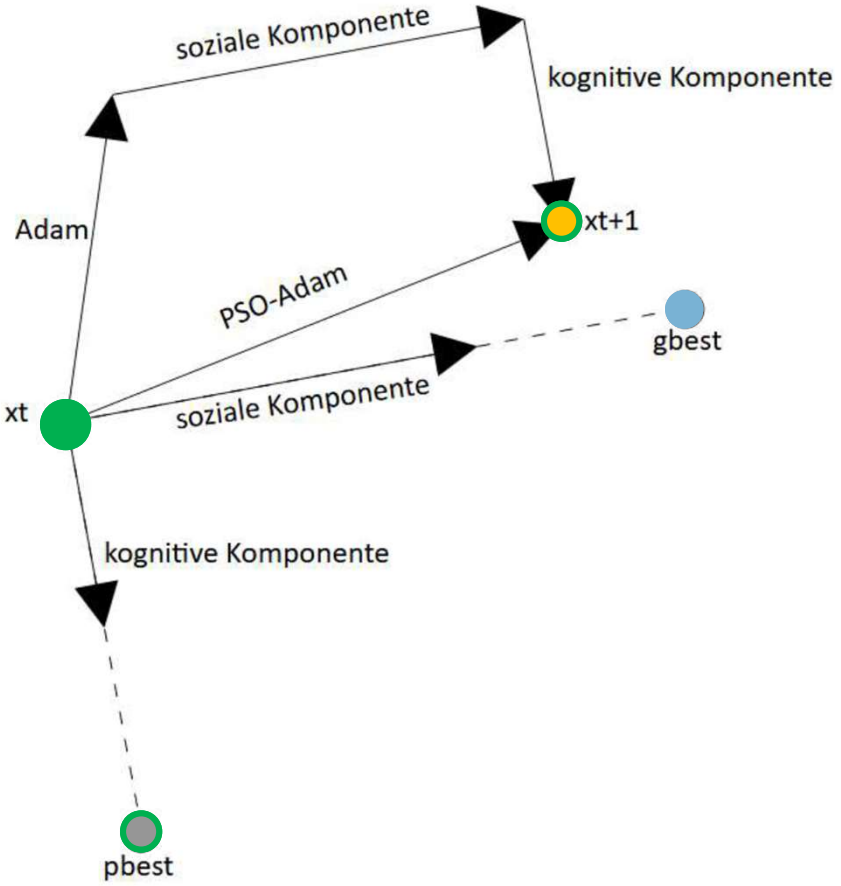
3. PSO-Adam – Vergleich zu Adam und PSO

Adam-Perspektive:

- Adam-Agenten werden aus ihrem Umfeld in Richtung besserer Parameter bewegt

PSO-Perspektive:

- Adam übernimmt die Rolle des Trägheitsterms (Momentum der Agenten)



4. IMPLEMENTIERUNG

In Tensorflow mit Keras

4. Implementierung – Tensorflow und Keras

- Open-Source-Framework für maschinelles Lernen
- Keras: High-level API in Tensorflow

Model():
- Klasse für KNN-Modelle
- implementiert die ersten 3 Schritte des Trainingszyklus
- steuert den Trainingsablauf mit fit() und trainstep()
- gibt Gradienten an Optimizer() weiter

Optimizer():
- Klasse für Optimierungsalgorithmen
- apply_gradients() führt Parameter-Update durch

Callback():
- Ermöglicht Funktionsaufrufe für das Aufzeichnen von Werten, Ändern von Parametern, Trainingsabbruch und weitere Eingriffe

4. Implementierung – Ziele der Implementierung

1. Funktionalität von PSO-Adam
   - Verwaltung der Adam-Agenten
   - Durchführung der PSO-Updates
   - Abrufbarkeit des besten Ergebnisses des Schwarms

2. Kompatibilität mit der Keras API
   - Nutzung der Keras-eigenen Funktionen
   - Intuitive Nutzbarkeit

3. Liefern von erwarteten Ergebnissen
   - Performanz mit PSO-Koeffizienten c_1, $c_2 = 0$

4. Implementierung – 1. Version

PSOAdamOptimizer()

PSOAdamCallback()

- Optimizer-Klasse verwaltet Adam-Agenten
  - Jeder Agent beinhaltet eine Model- und eine Adam-Optimizer-Instanz
- Steuerung des Trainings wird durch die Callback-Klasse übernommen
- Bestes Ergebnis in Standard-Model-Instanz gespeichert

ABER:

- Model-Klasse gibt nur ermittelte Gradienten des Hauptmodells an den Optimizer weiter
- Zugriff auf die Trainingsdaten wird nötig

➢ Ansatz ist unintuitiv, schlecht performant und liefert unerwartete Ergebnisse

4. Implementierung – 2. Version

PSOAdamOptimizer()

PSOModel()

- Übertragung der Funktionalitäten des Callbacks in die Model-Klasse
- PSOModel gibt Loss an PSO-Update weiter
- Funktionalität der Optimizer-Klasse bleibt weitesgehend gleich
  - Beinhaltet Agenten und Modelle sowie Funktionen für PSO

ABER:

- Optimizer-Klasse beinhaltet nur Funktionen, die direkt über das PSOModel angesprochen werden
- Zugriff auf die Agentenmodelle über die Optimizer Klasse macht wenig Sinn

➢ Ansatz ist etwas intuitiver, kann aber vereinfacht und generalisiert werden

4. Implementierung – Finale Version: PSOModel()

PSOModel()

K Adam()

Optimizerklasse erweist sich als Redundant

– Überführung aller Funktionen nach PSOModel()

1. PSOModel mit gewünschter Architektur initialisieren
2. Agentenmodelle an das PSOModel übergeben
3. Optimizer auswählen (z.B. Adam() aus Keras)
4. Training starten

➢ Kombination von PSOModel() mit beliebigen Keras Optimizer-Klassen

5. EXPERIMENTE

Aufbau und Auswertung

5. Experimente – Aufbau

Genutzte Feedforward Netzwerke:

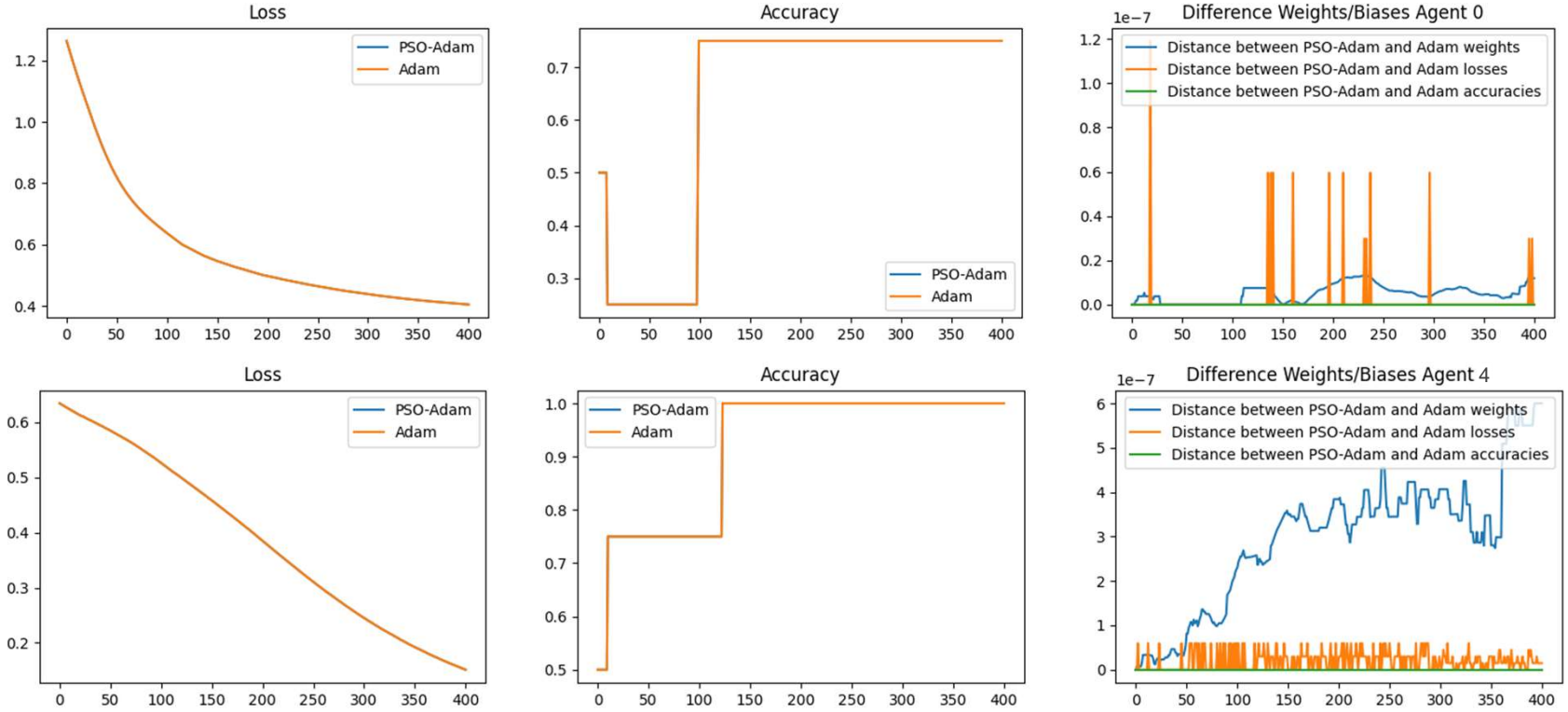
Experiment	Datensatz	Architektur	Aktivierungsfunktionen
XOR	XOR: (00, 0), (01, 1), (10, 1), (11, 0)	[2, 2, 1]	[- , ReLU, Sigmoid]
Tiny	MNIST, Fashion MNIST	[784, 8, 10]	[- , ReLU, Softmax]
Small		[784, 100, 10, 10]	[- , ReLU, ReLU, Softmax]
Large		[784, 100, 100, 10]	[- , ReLU, ReLU, Softmax]

- 10 Agenten
- Vergleich mit Adam() aus Keras
- mit PSO-Koeffizienten = 0
- Performanz-Experimente

5. Experimente – Erwartungen an Vergleich mit Adam() aus Keras

- Die Implementierung von PSO-Adam nutzt die Adam-Version aus Keras.
- Ohne PSO-Bewegungen sollten sich die Agenten von PSO-Adam so verhalten wie als wären sie nicht Teil eines Schwarms.
- Vergleich mit Adam() auf den gleichen XOR-Modellen

5. Experimente – Ergebnisse des Vergleichs mit Adam() aus Keras



– Minimale Unterschiede durch Rundungsfehler, sonst Deckungsgleich

5. Experimente – Erwartungen für die Performanzexperimente

- Soziale Komponente skaliert direkt mit dem Abstand zwischen Agenten

$$c_1 \cdot r_1 \cdot (pbest_i - x_{i,t})$$

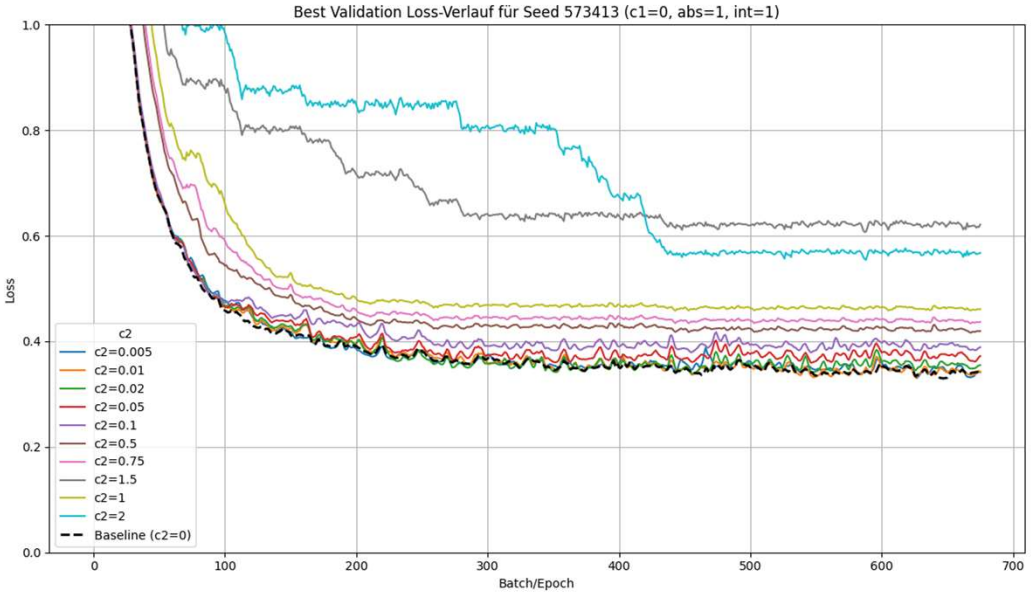
- Kognitive Komponente skaliert mit Sozialer Komponente und Adam

$$c_2 \cdot r_2 \cdot (gbest - x_{i,t})$$

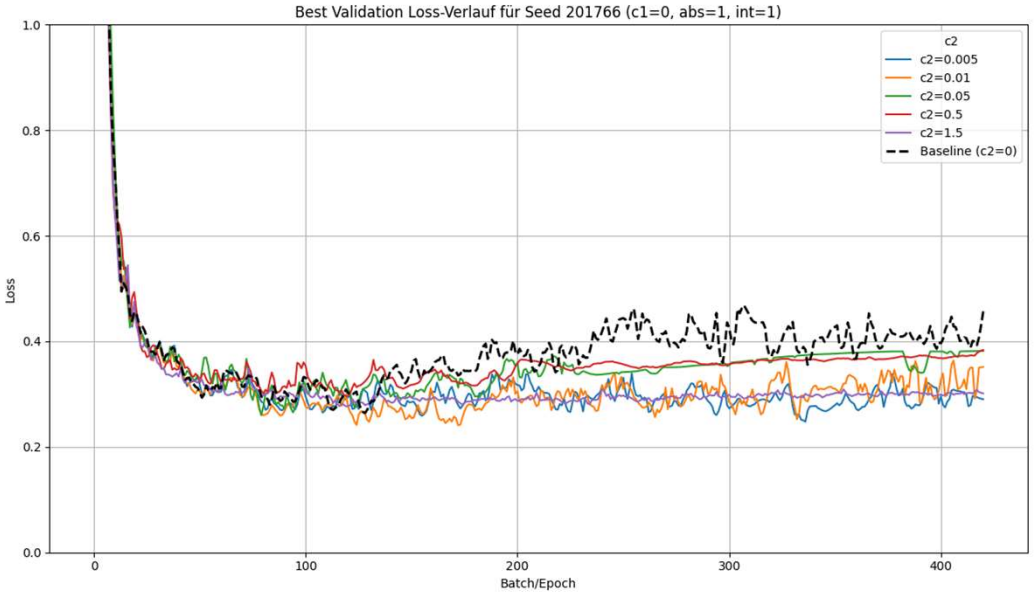
Vermutung:

- c_2 abhängig von Architektur
  - Standardwerte zu groß
  - zu starke Dominanz bei zu großen Werten
- c_1 Standardwerte sollten optimal sein
  - kein Einfluss bei zu kleinen Werten

5. Experimente – Ergebnisse

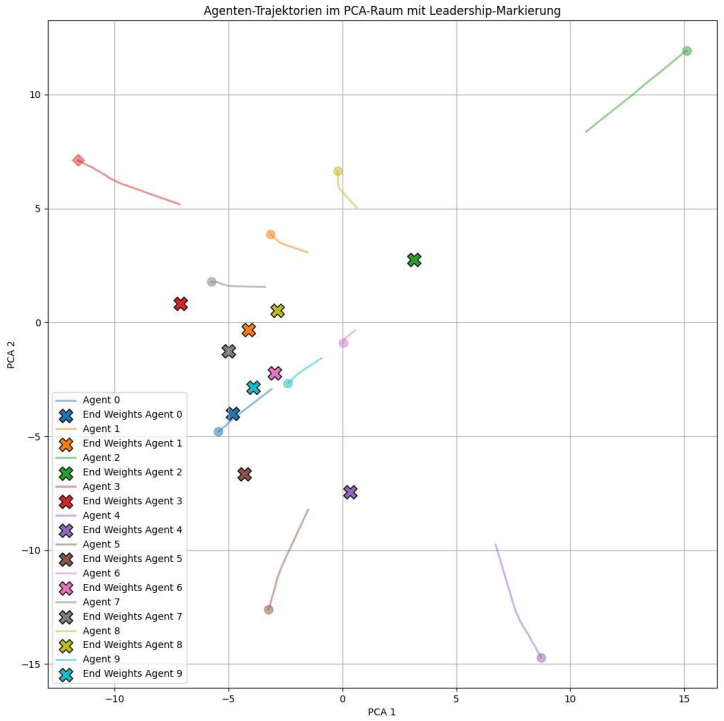


Lossverläufe für Tiny mit $c_1 = 0$

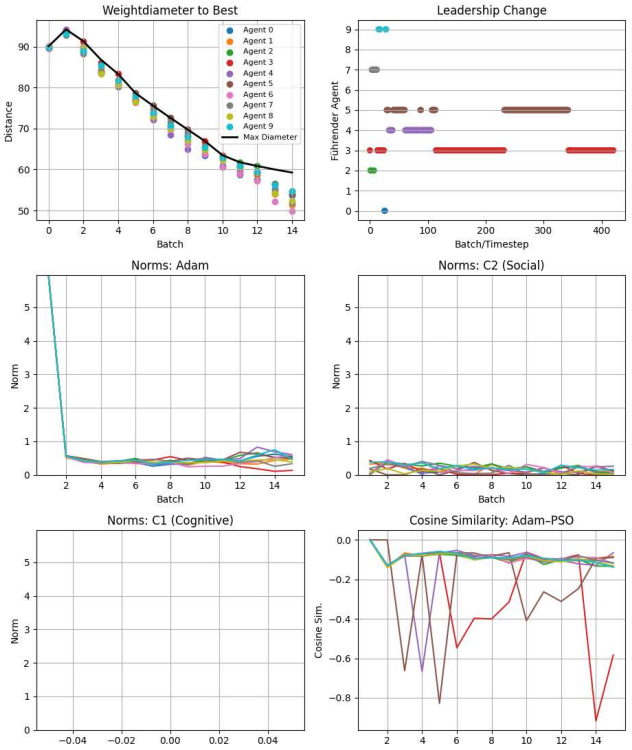


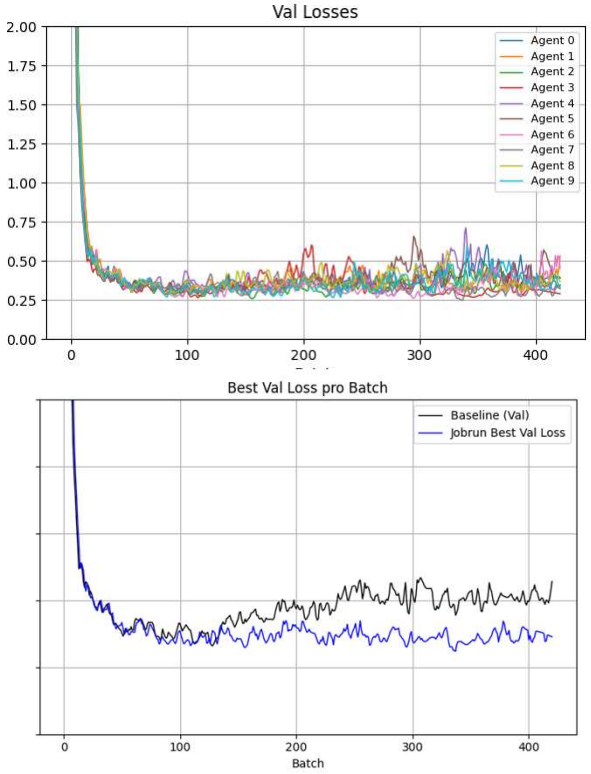
Lossverläufe für Large mit $c_1 = 0$

5. Experimente – Ergebnisse



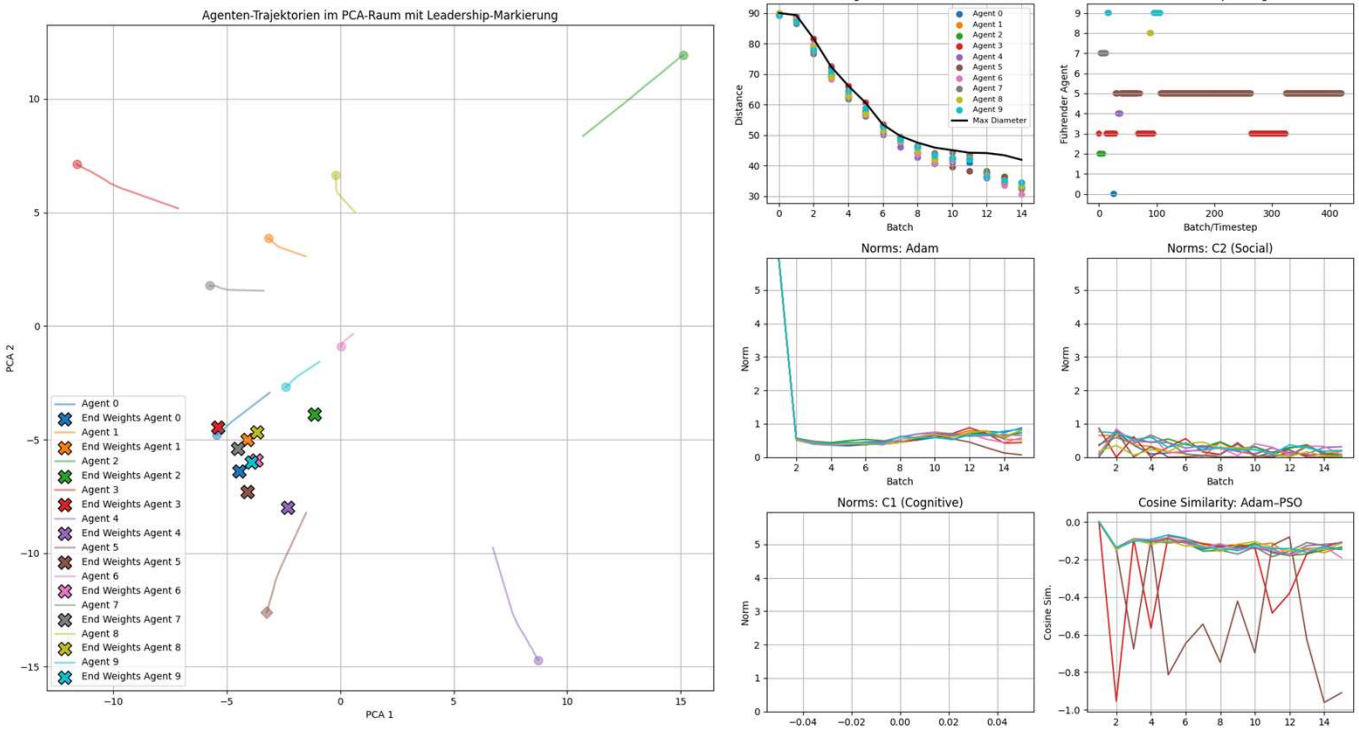
2D PCA für Large mit $c_2 = 0.005$



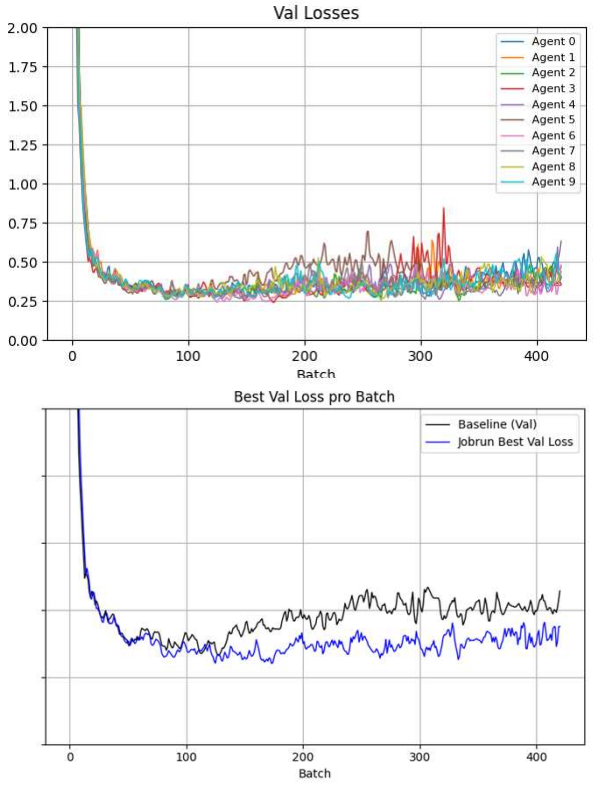


Lossverläufe für Large mit $c_2 = 0.005$

5. Experimente – Ergebnisse

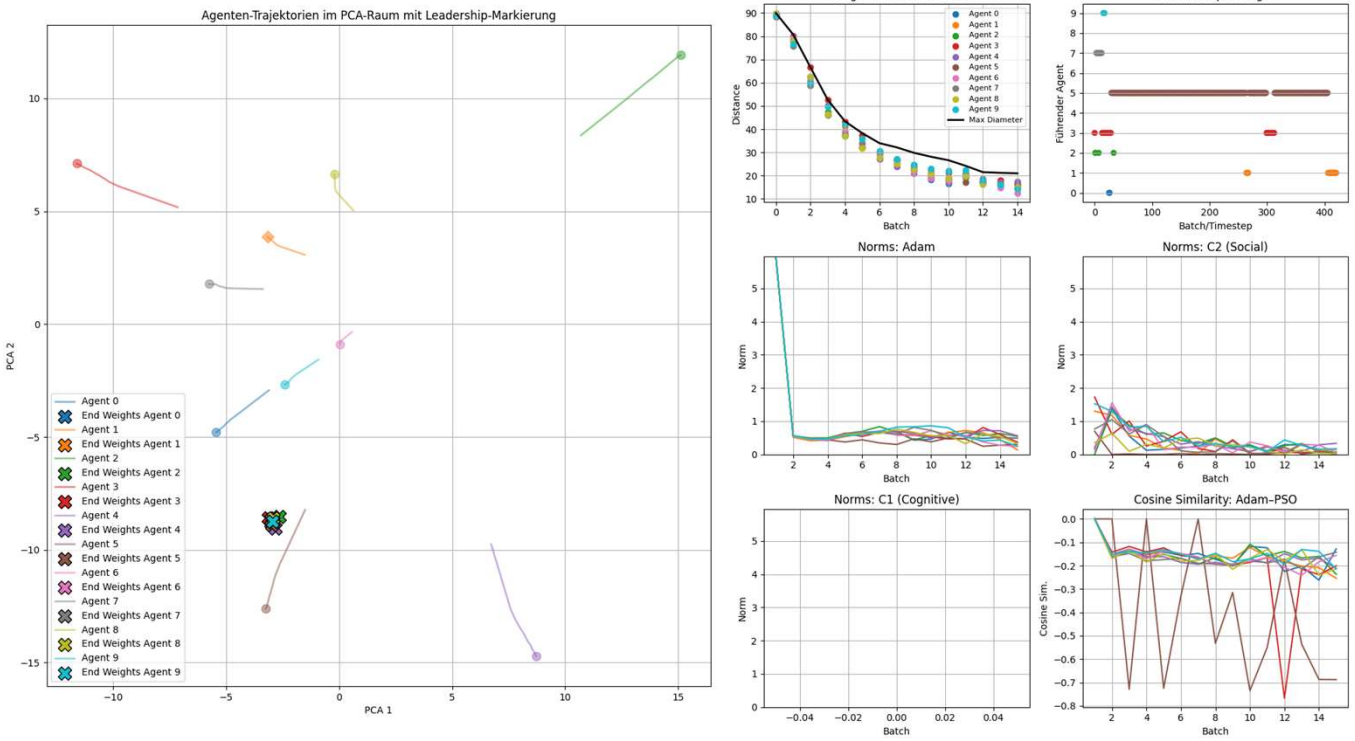


2D PCA für Large mit $c_2 = 0.01$

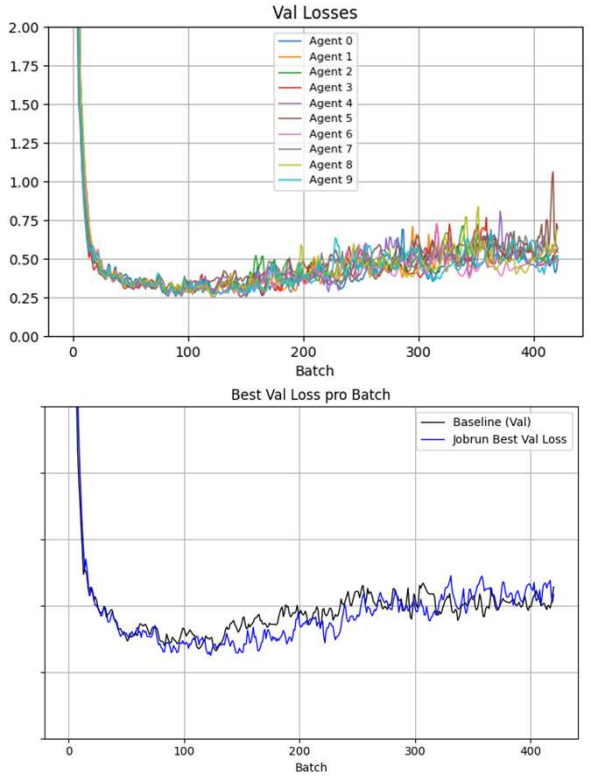


Lossverläufe für Large mit $c_2 = 0.01$

5. Experimente – Ergebnisse

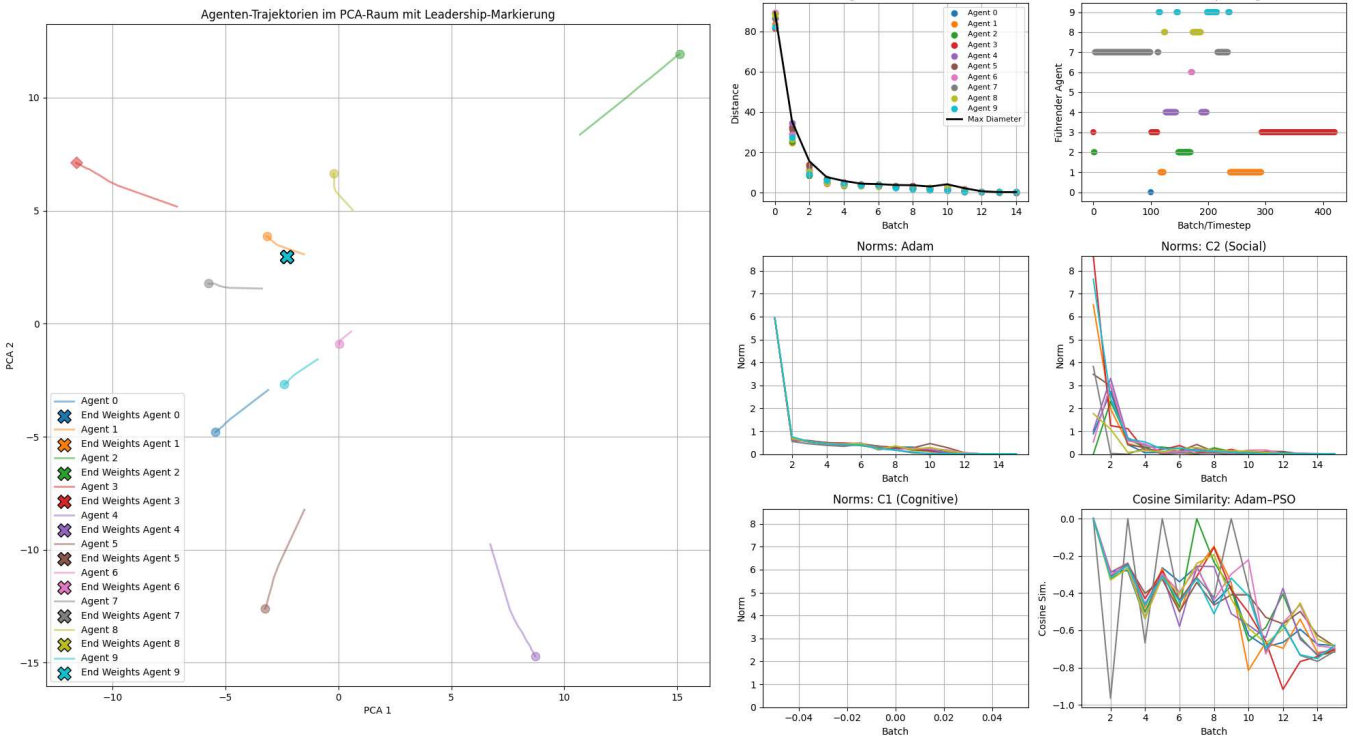


2D PCA für Large mit $c_2 = 0.02$

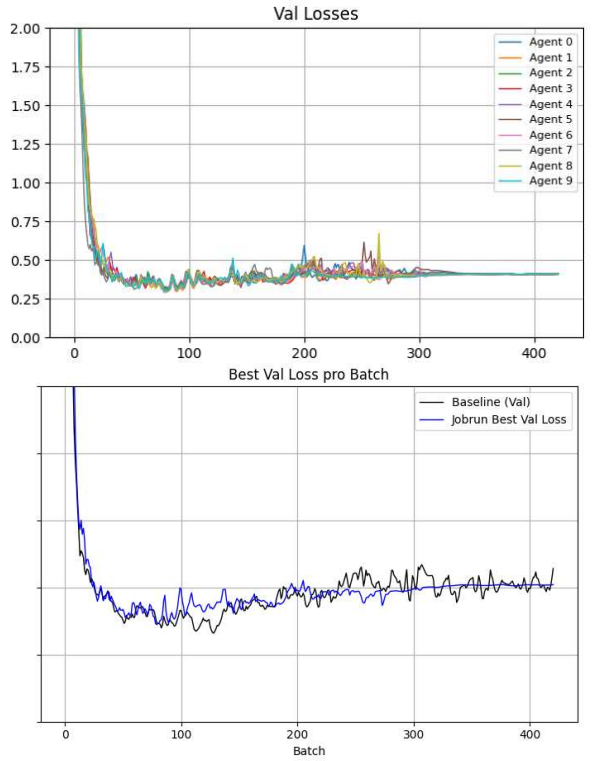


Lossverläufe für Large mit $c_2 = 0.02$

5. Experimente – Ergebnisse

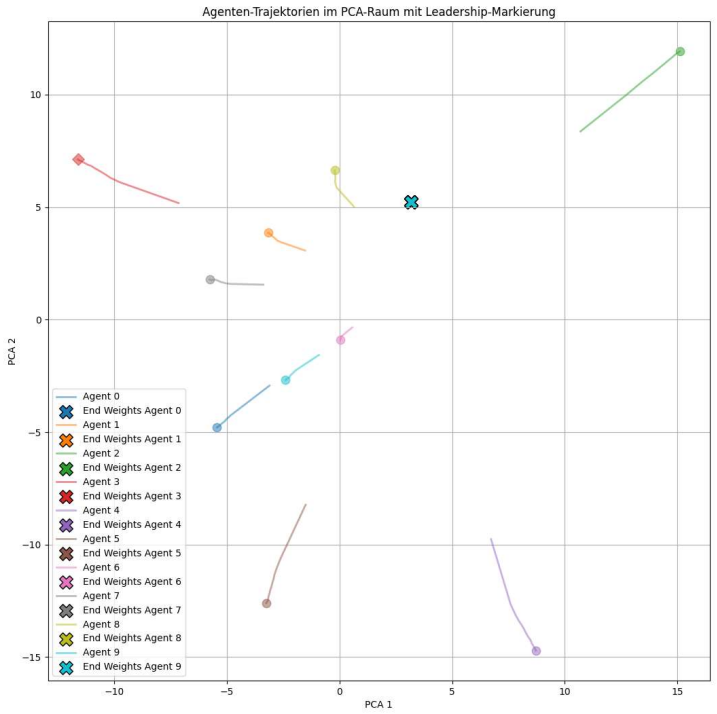


2D PCA für Large mit $c_2 = 0.1$

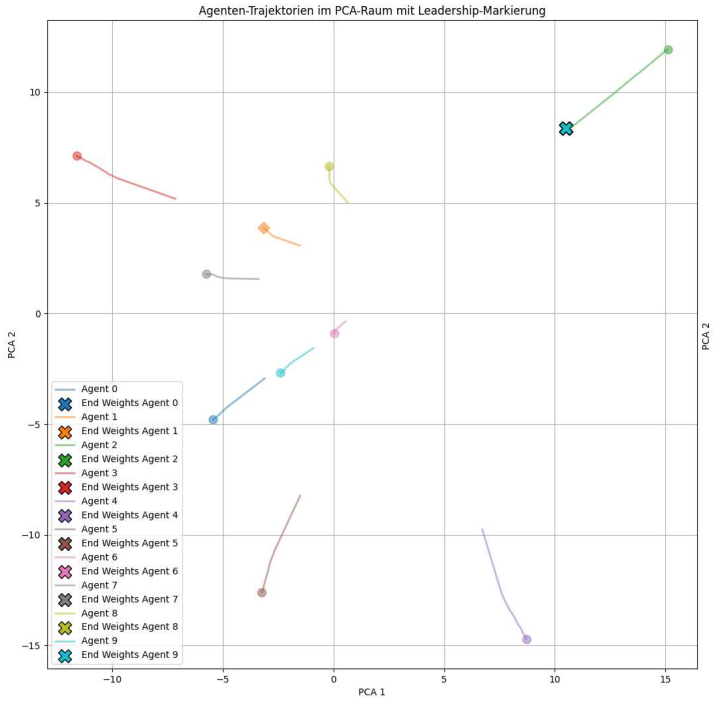


Lossverläufe für Large mit $c_2 = 0.1$

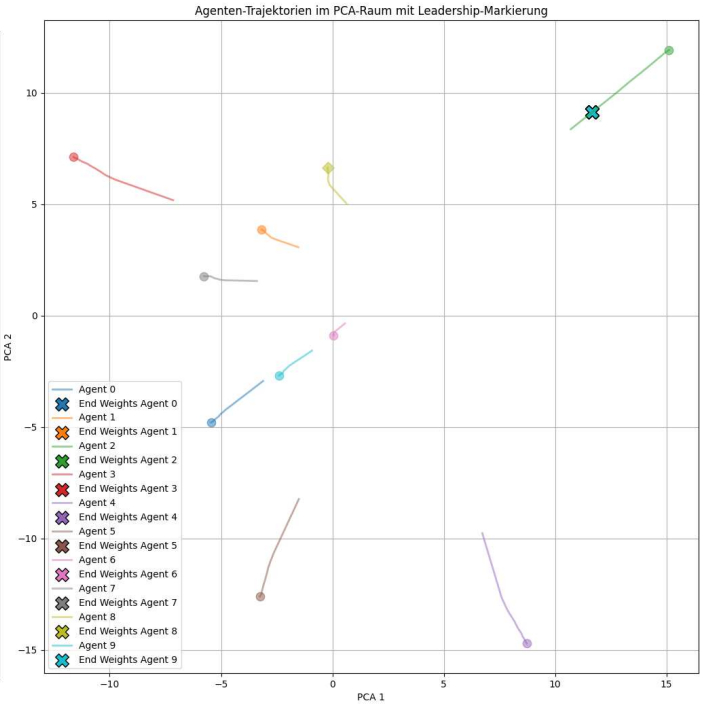
5. Experimente – Ergebnisse



2D PCA für Large mit $c_2 = 0.5$

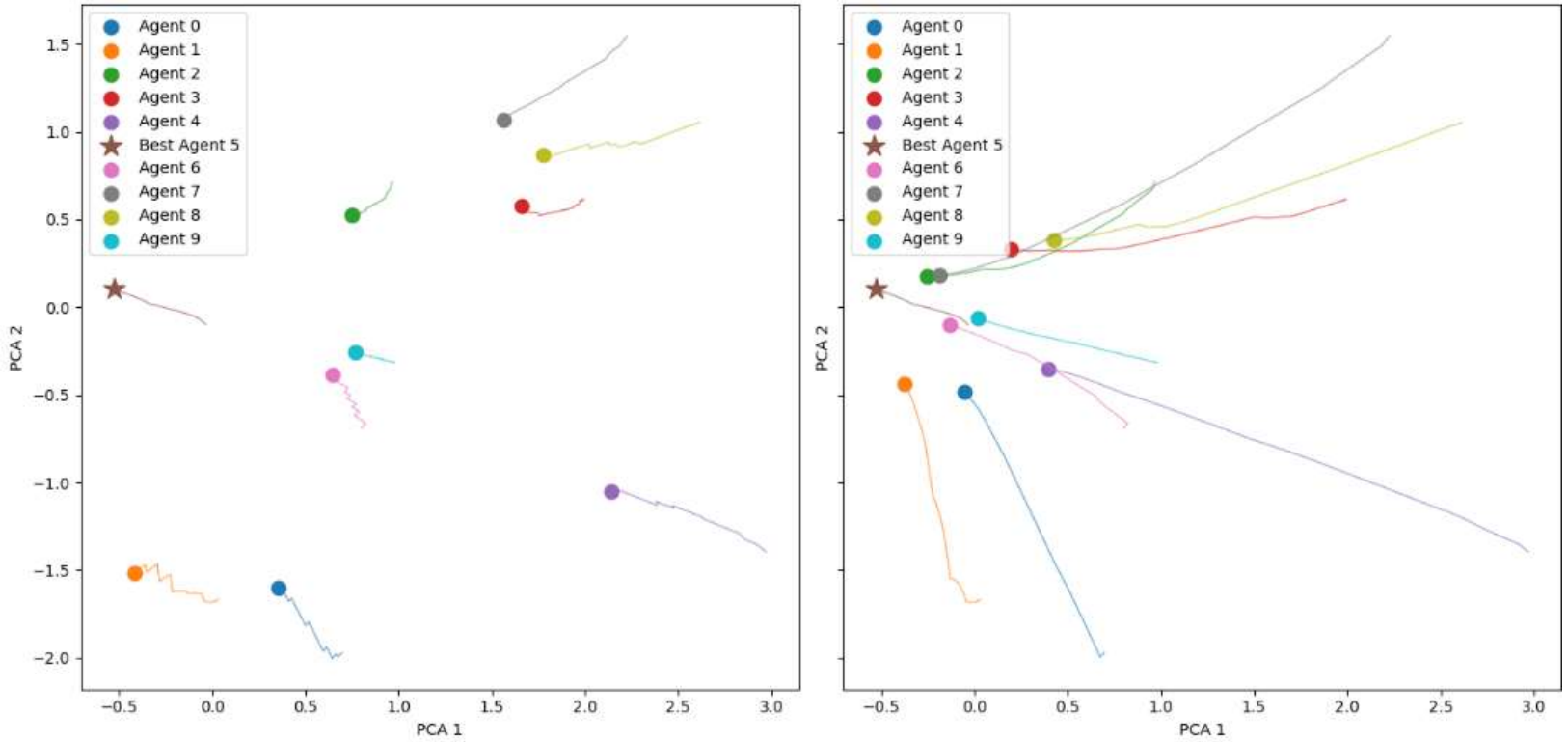


2D PCA für Large mit $c_2 = 1.5$



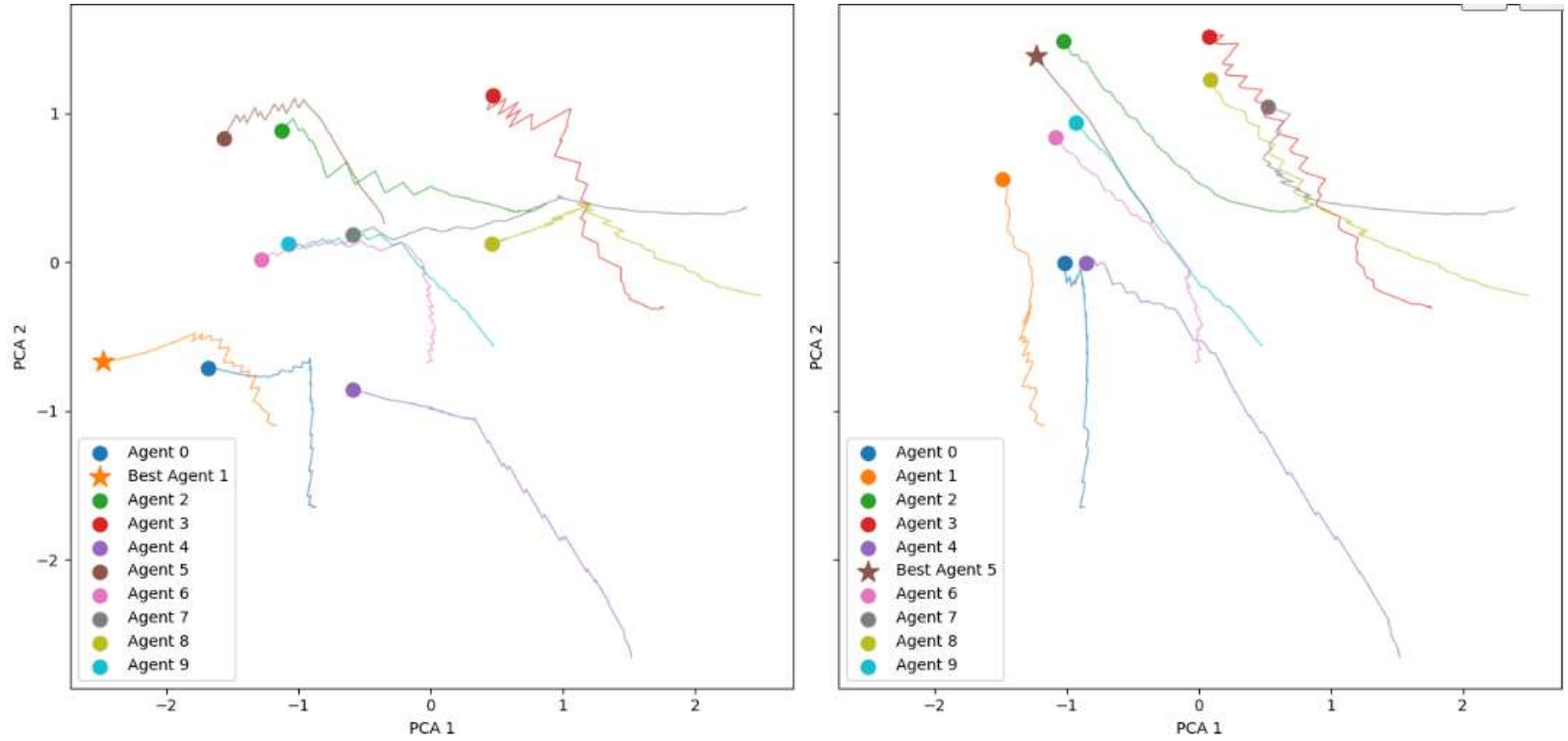
2D PCA für Large mit $c_2 = 1.5$

5. Experimente – Ergebnisse



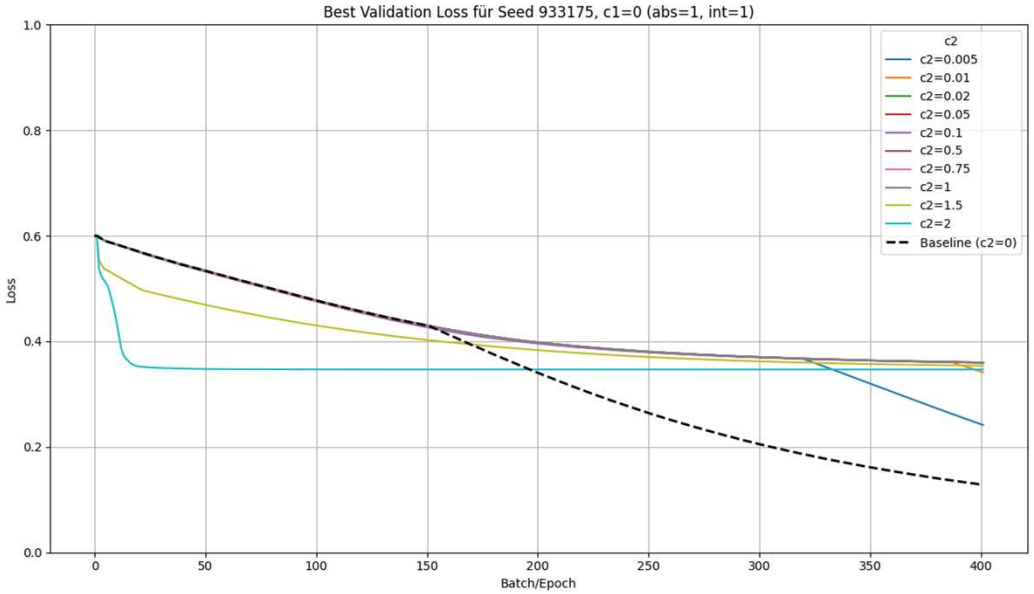
2D PCA für XOR, Links kleines c2 rechts großes c2

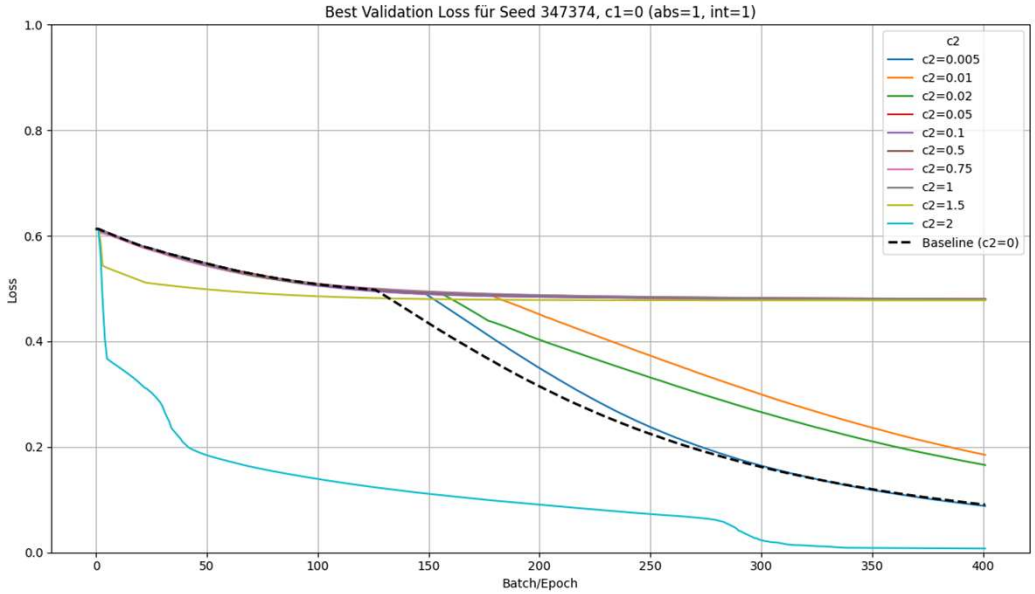
5. Experimente – Ergebnisse



2D PCA für XOR, Links kleines c2 rechts großes c2

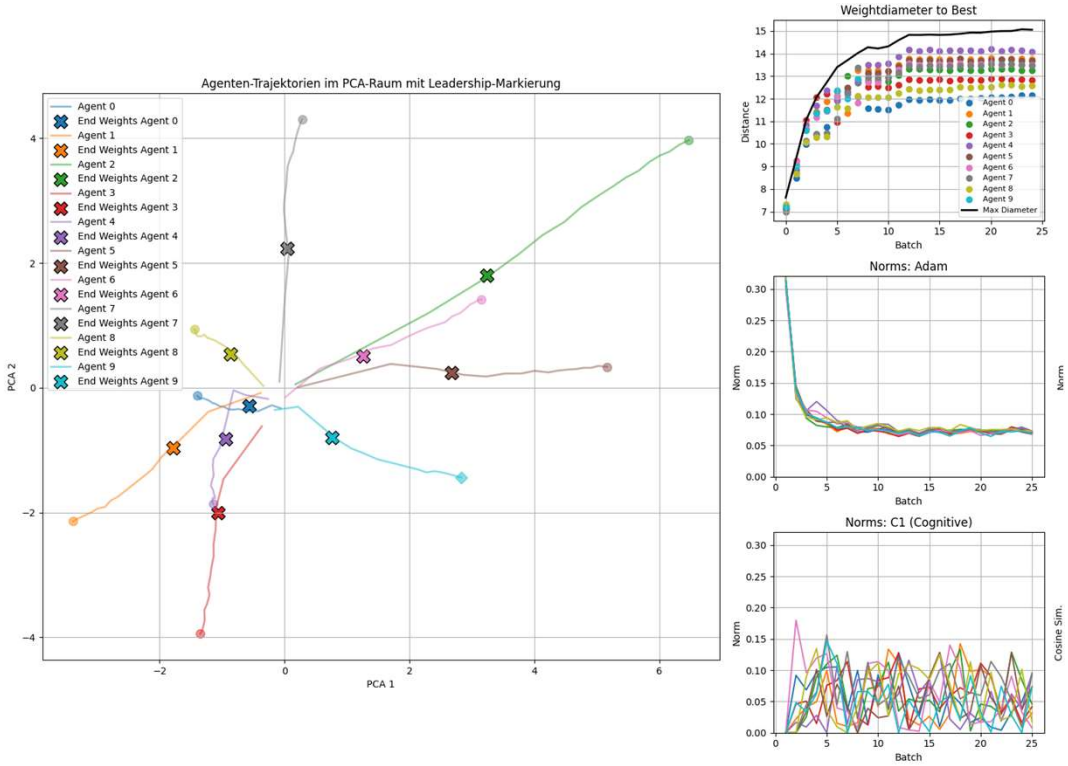
5. Experimente – Ergebnisse



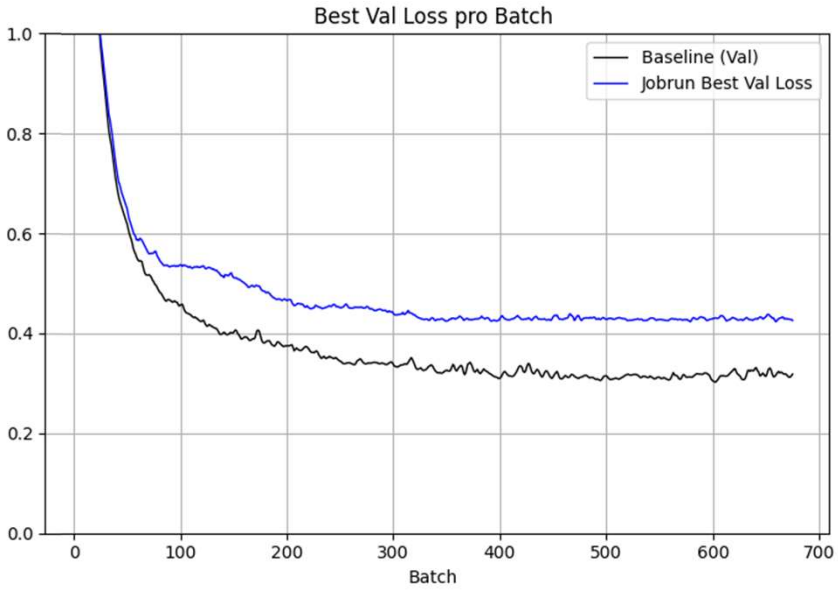


Lossverläufe für XOR mit $c_1 = 0$

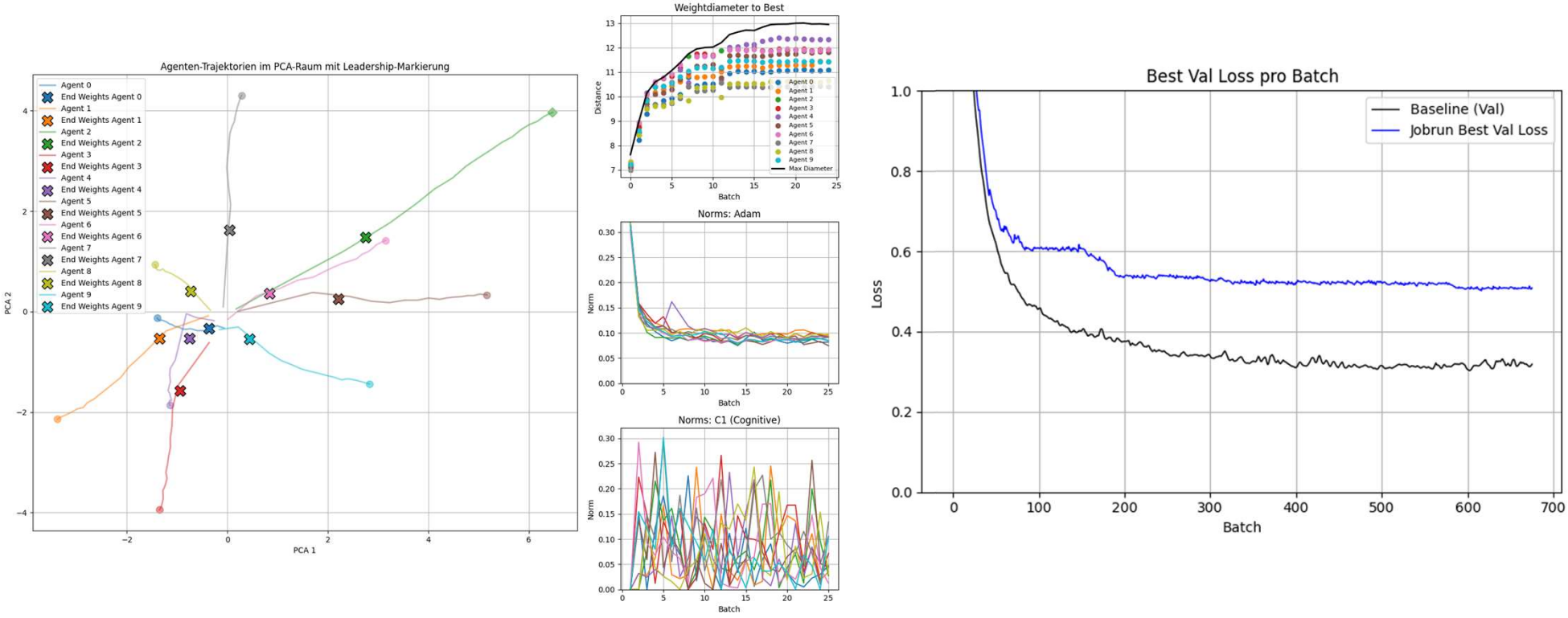
5. Experimente – Ergebnisse Einfluss von c_1



2D PCA für Tiny mit $c_1 = 0.5$



5. Experimente – Ergebnisse Einfluss von c_1



2D PCA für Tiny mit $c_1 = 1.5$

5. Experimente – Ergebnisse

- Performanz von PSO-Adam im jetzigen Zustand eher durchwachsen

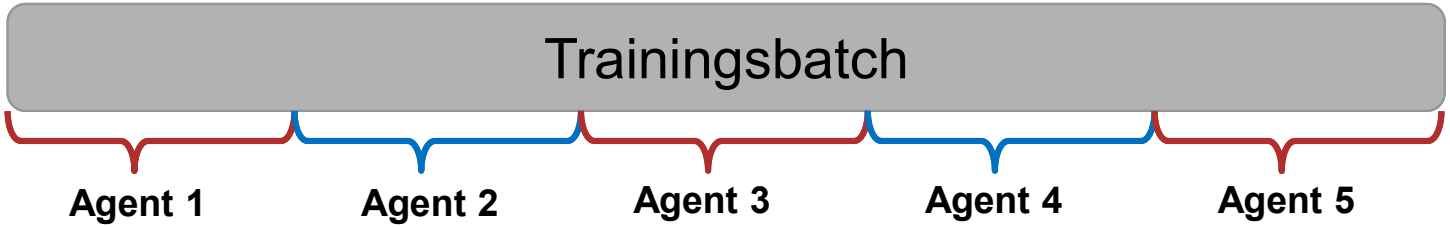
- c_1 allein verlangsamt das Lernen
- Großes c_2 kann zu zufälligen Lernerfolgen führen
  - Wahrscheinlicher bei sehr kleinen Netzen
- Agenten, die schnell lokale Minima finden hindern den Schwarm an Exploration

6. ERWEITERUNGEN

6. Erweiterungen – Implementierte zusätzliche Parameter

Agentbatchsize:

- Aufteilung des Trainingsbatches auf die Adam Agenten
- Bsp.: 5 Adam Agenten und Agentbatchsize = Batchsize / 5

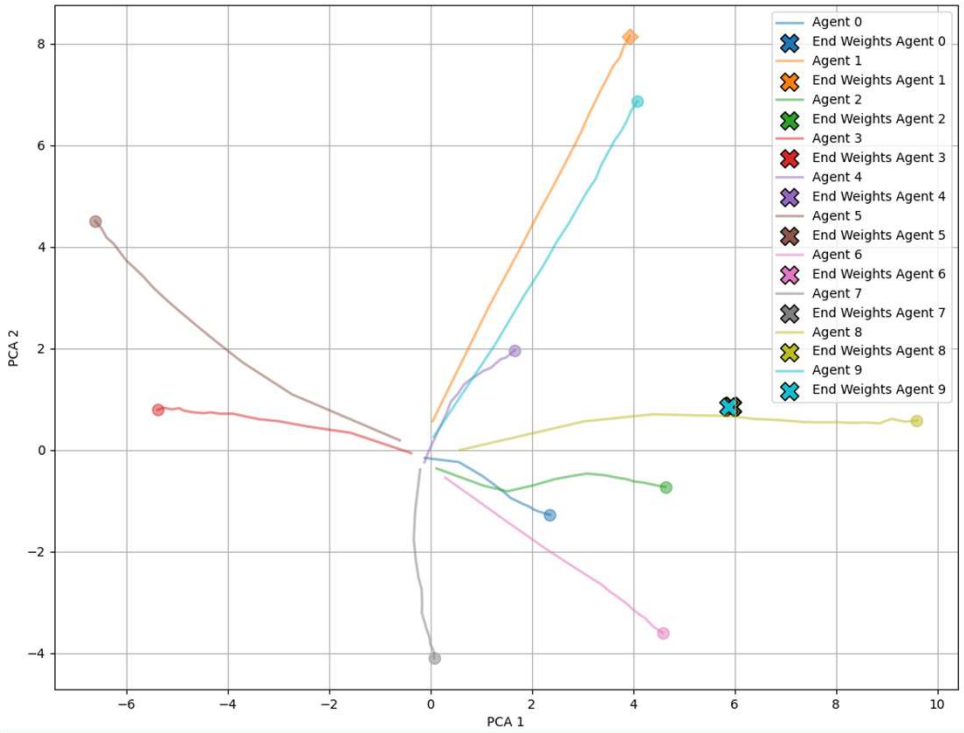


PSO-Intervall:

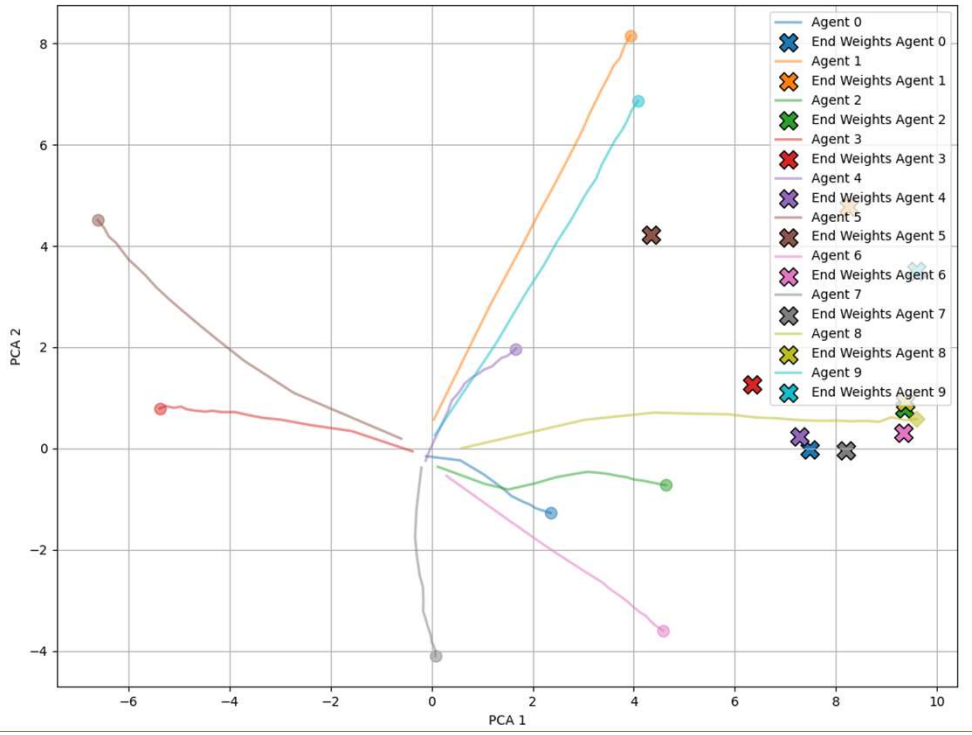
- PSO pausiert für festgelegte Intervalle
- Agenten Explorieren mit Adam-Schritten, bevor sie vom Global Best angezogen werden

6. Erweiterungen – Ergebnisse für PSO-Intervall

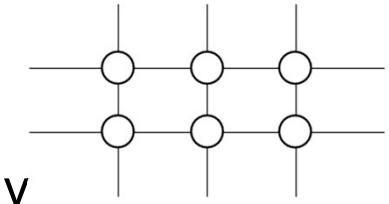
PSO-Interval = 1



PSO-Interval = 20

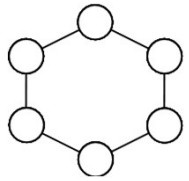


6. Erweiterungen – weitere Möglichleiten



PSO-Nachbarschaften:

- Radius Basierte Nachbarschaften vermutlich zu Rechenintensiv
- Kleinere Nachbarschaften wie Ring oder von-Neumann entschleunigen PSO
  - ➢ mehr lokale Exploration



Integration von Velocity und Inertia

- PSO-Bewegung ist bei PSO-Adam nur eine ruckartige Bewegung
- In Kombination mit PSO-Intervall:
  - ➢ Agent wird in eine Richtung angestoßen und bremst gleichmäßig ab

Maßnahmen zur PSO Regulierung

- Soziale Komponente durch Loss skalieren lassen

7. DISKUSSION

Zeit für Fragen

UNIVERSITÄT
LEIPZIG

VIELEN DANK!