

## L06: Embeddings & RL

Deep Dive: Theory, Implementation, and Applications

### Methods and Algorithms

MSc Data Science

Spring 2026

# Outline

1 Problem

2 Method

3 Solution

4 Practice

5 Decision Framework

6 Summary

## The Problem with One-Hot Encoding

- Vocabulary of 10,000 words → 10,000-dim sparse vectors
- No semantic similarity: “king” and “queen” equally distant from “car”
- Curse of dimensionality

## Solution: Dense Embeddings

- Map words to dense vectors (50-300 dimensions)
- Similar words → similar vectors
- Learn from context (distributional hypothesis)

---

“You shall know a word by the company it keeps” – Firth, 1957

## Word2Vec: Skip-gram

**Objective:** Predict context words given target word

$$P(w_{context} | w_{target}) = \frac{\exp(v_{context}^T v_{target})}{\sum_{w \in V} \exp(v_w^T v_{target})}$$

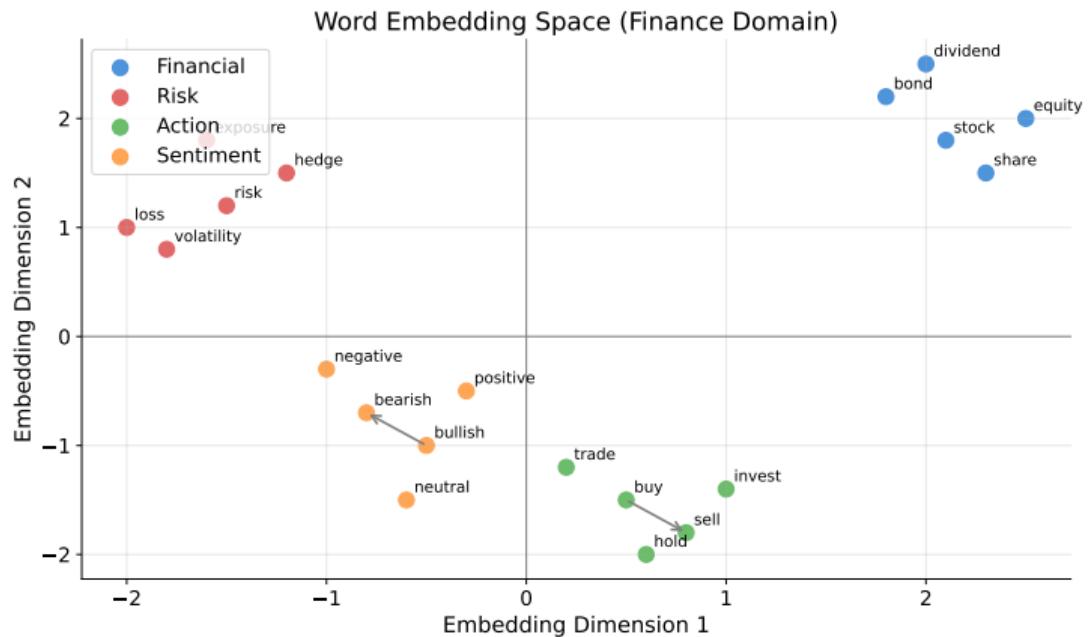
**Training:**

- Slide window over text corpus
- For each word, predict surrounding words
- Update embeddings via gradient descent

---

Skip-gram works well for rare words; CBOW better for frequent words

# Word Embedding Space



[https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L06\\_EMBEDDINGS\\_RL/01\\_word\\_embedding\\_space](https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L06_EMBEDDINGS_RL/01_word_embedding_space)

Finance terms cluster by semantic category in embedding space

# Word Analogies

## Famous Example:

$$\vec{king} - \vec{man} + \vec{woman} \approx \vec{queen}$$

## Finance Examples:

- $\vec{stock} - \vec{equity} + \vec{debt} \approx \vec{bond}$
- $\vec{CEO} - \vec{company} + \vec{country} \approx \vec{president}$

## How it works:

- Vector arithmetic in embedding space
- Relationships encoded as directions

---

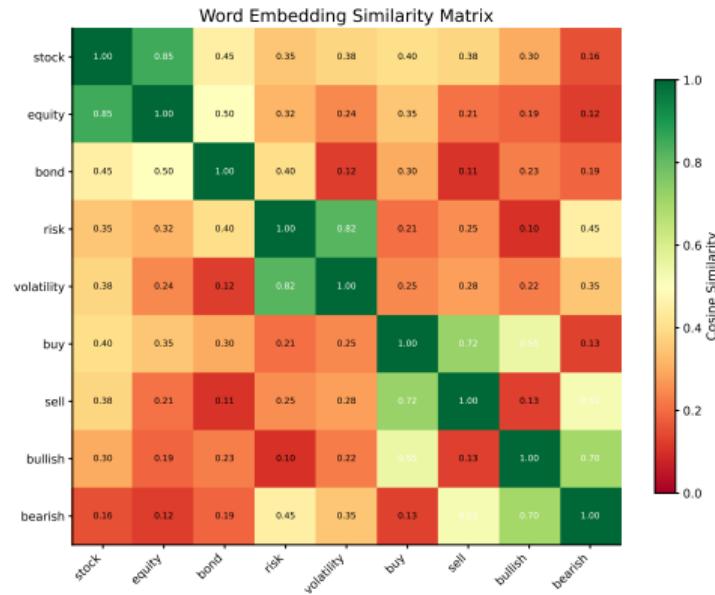
Embeddings capture relational structure, not just similarity

# Similarity Measures

## Cosine Similarity:

$$\text{sim}(u, v) = \frac{u \cdot v}{\|u\| \cdot \|v\|} = \cos(\theta)$$

- Range:  $[-1, 1]$ ; 1=identical, 0=orthogonal, -1=opposite



Cosine similarity ignores magnitude, focuses on direction

## Popular Options:

- **Word2Vec**: Google, 300-dim, 3M words
- **GloVe**: Stanford, trained on Wikipedia + Common Crawl
- **FastText**: Facebook, handles subwords (OOV robust)

## Domain-Specific:

- FinBERT: Pre-trained on financial text
- BioBERT: Biomedical domain

---

Fine-tuning pre-trained embeddings usually outperforms training from scratch

## Applications:

- **Sentiment Analysis:** News → embedding → positive/negative
- **Document Similarity:** Find similar SEC filings
- **Named Entity Recognition:** Extract company names
- **Event Detection:** Identify earnings announcements

## Sentence Embeddings:

- Average word vectors (simple baseline)
- Doc2Vec (paragraph vectors)
- Sentence-BERT (state-of-the-art)

---

Aggregate word embeddings to represent documents

## Part 2: Reinforcement Learning Framework

### Key Components:

- **Agent**: Learner/decision-maker
- **Environment**: What agent interacts with
- **State  $s$** : Current situation
- **Action  $a$** : What agent can do
- **Reward  $r$** : Feedback signal

Reinforcement Learning: Agent-Environment Interaction



At each time step  $t$ :

Agent observes state, takes action, receives reward

[https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/l06\\_EMBEDDINGS\\_RL/03\\_rl\\_loop](https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/l06_EMBEDDINGS_RL/03_rl_loop)

RL: Learning from interaction, not from labeled examples

**MDP Tuple:**  $(S, A, P, R, \gamma)$

- $S$ : Set of states
- $A$ : Set of actions
- $P(s'|s, a)$ : Transition probability
- $R(s, a, s')$ : Reward function
- $\gamma \in [0, 1]$ : Discount factor

**Markov Property:**

$$P(s_{t+1}|s_t, a_t, s_{t-1}, \dots) = P(s_{t+1}|s_t, a_t)$$

---

Future depends only on current state, not history

# Policy and Value Functions

**Policy:**  $\pi(a|s) = P(A_t = a|S_t = s)$

- Maps states to action probabilities
- Goal: Find optimal policy  $\pi^*$

**Value Function:**

$$V^\pi(s) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s \right]$$

**Q-Function (Action-Value):**

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s, A_0 = a \right]$$

---

**Q-function:** expected return starting from state  $s$ , taking action  $a$

## Optimal Q-Function:

$$Q^*(s, a) = \mathbb{E} \left[ R + \gamma \max_{a'} Q^*(s', a') \right]$$

## Interpretation:

- Value = immediate reward + discounted future value
- Recursive definition enables dynamic programming

## Optimal Policy:

$$\pi^*(s) = \arg \max_a Q^*(s, a)$$

---

Bellman equation: foundation of all value-based RL methods

## Update Rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

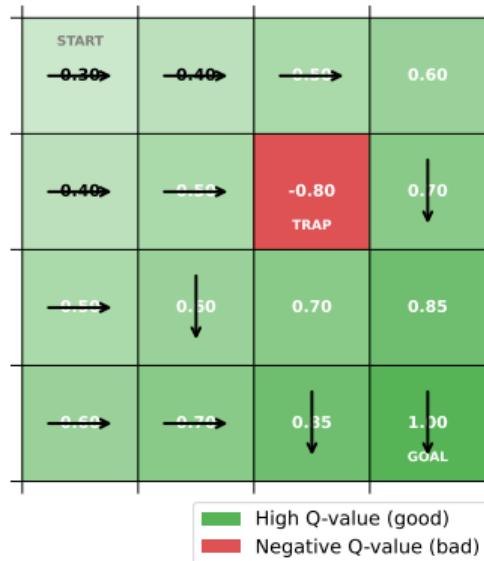
## Algorithm:

1. Initialize  $Q(s, a)$  arbitrarily
2. For each episode:
  - Observe state  $s$
  - Choose action  $a$  ( $\epsilon$ -greedy)
  - Execute  $a$ , observe  $r, s'$
  - Update  $Q(s, a)$

---

Q-learning is off-policy: learns optimal Q regardless of policy followed

## Q-Learning: Grid World with Learned Q-Values



[https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/l06\\_EMBEDDINGS\\_RL/04\\_q\\_learning\\_grid](https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/l06_EMBEDDINGS_RL/04_q_learning_grid)

Arrows show policy; colors show Q-values (green=high, red=negative)

## The Dilemma:

- **Exploit:** Choose best known action (greedy)
- **Explore:** Try new actions (discover better options)

## $\epsilon$ -Greedy Strategy:

$$a = \begin{cases} \arg \max_a Q(s, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

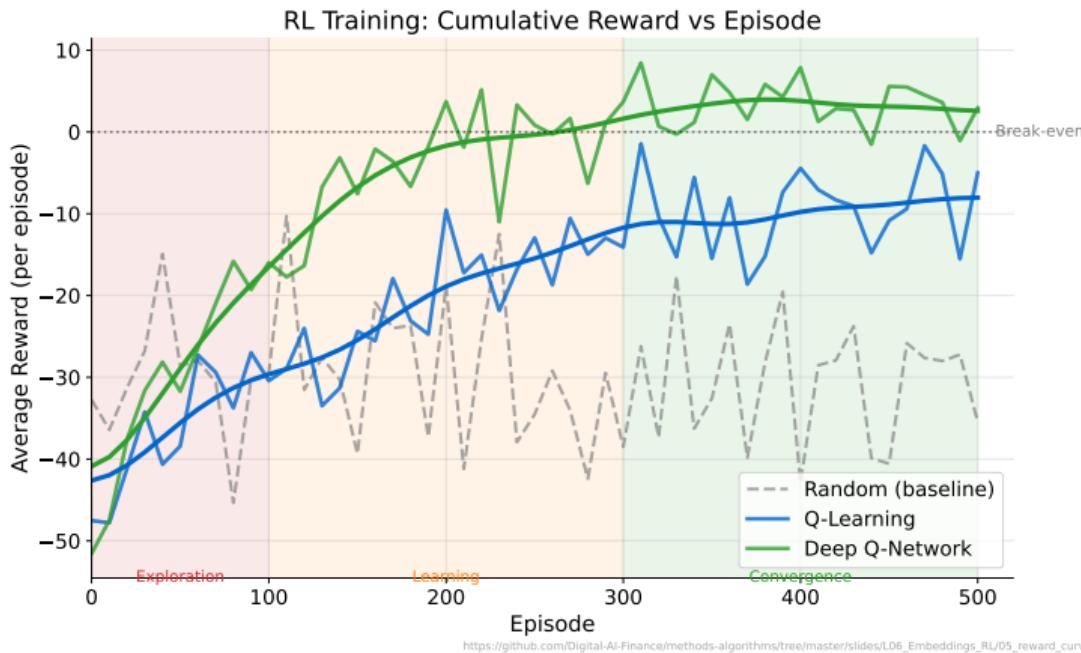
## Decay Schedule:

- Start with high  $\epsilon$  (explore more)
- Decay  $\epsilon$  over time (exploit more)

---

Balance: too much exploration wastes time; too little misses optima

# Learning Curves



Reward improves as agent learns; DQN often outperforms tabular Q-learning

### Formulation:

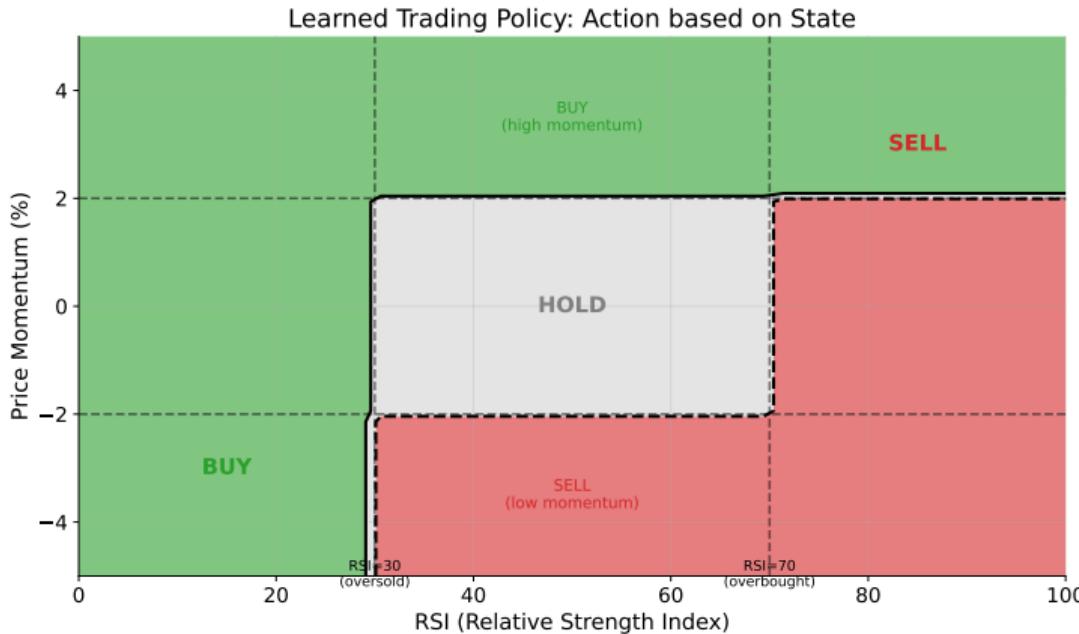
- **State:** Price history, portfolio, technical indicators
- **Action:** Buy, sell, hold (+ position size)
- **Reward:** Profit/loss, risk-adjusted return

### Challenges:

- Non-stationary environment
- High noise, low signal-to-noise ratio
- Transaction costs
- Partial observability

---

RL for trading is active research area; not solved problem



Learned policy: buy when oversold/high momentum, sell when overbought

**Idea:** Use neural network to approximate Q-function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

## Key Innovations:

- **Experience Replay:** Store transitions, sample randomly
- **Target Network:** Separate network for stability
- **Function Approximation:** Handle large state spaces

---

DQN: breakthrough for RL on Atari games (Mnih et al., 2015)

## Direct Policy Optimization:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) \cdot Q^{\pi_{\theta}}(s, a)]$$

## Algorithms:

- REINFORCE (vanilla policy gradient)
- Actor-Critic (combine value and policy)
- PPO (Proximal Policy Optimization)
- A3C (Asynchronous Advantage Actor-Critic)

---

Policy gradient: directly optimize policy, can handle continuous actions

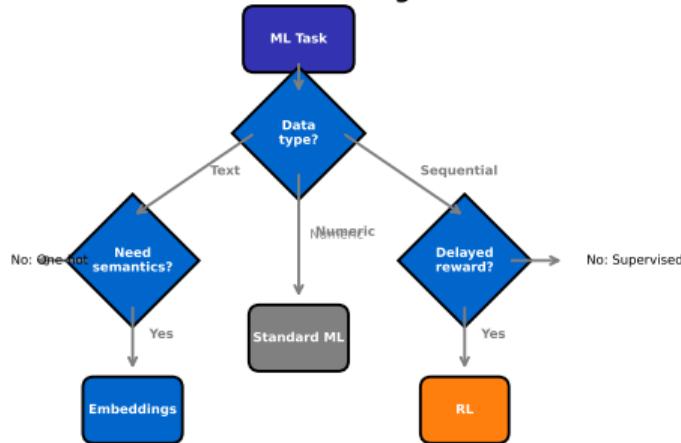
## Open the Colab Notebook

- Exercise 1: Explore word embeddings with Word2Vec
- Exercise 2: Implement basic Q-learning
- Exercise 3: Apply RL to a simple trading environment

Link: <https://colab.research.google.com/> [TBD]

## Part 4: When to Use What

### When to Use Embeddings vs RL



*Embeddings: Text, categorical -> dense vectors (Word2Vec, BERT)*

*RL: Sequential decisions with delayed rewards (trading, games)*

[https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L06\\_EMBEDDINGS\\_RL/07\\_decision\\_flowchart](https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L06_EMBEDDINGS_RL/07_decision_flowchart)

---

**Embeddings for text/categorical; RL for sequential decisions**

## Comparison Table

Aspect	Embeddings	RL
Input	Text, categorical	State sequence
Output	Dense vectors	Actions/policy
Learning	Unsupervised/supervised	Trial and error
Signal	Context (words)	Rewards
Key challenge	Semantics	Credit assignment
Finance use	Sentiment	Trading

---

Both transform complex inputs into learnable representations

## Part 5: Implementation

### Embeddings in Python:

- `gensim.models.Word2Vec`: Train your own
- `gensim.downloader.load('glove-wiki-gigaword-100')`: Pre-trained
- `transformers.BertModel`: BERT embeddings

### RL Libraries:

- `gymnasium`: Environment interface (formerly OpenAI Gym)
- `stable-baselines3`: Pre-implemented algorithms
- `ray[rllib]`: Scalable RL

---

Start with pre-trained embeddings; use stable-baselines3 for RL

## Embeddings:

- Start with pre-trained, fine-tune if needed
- Check domain match (general vs financial)
- Visualize with t-SNE/UMAP to verify quality

## RL:

- Start simple (tabular Q-learning before DQN)
- Reward shaping is crucial (sparse rewards are hard)
- Normalize observations
- Use established environments first (Gym, FinRL)

---

Both domains: start simple, iterate, validate thoroughly

## Embeddings:

- Dense vector representations of text/categories
- Capture semantic similarity
- Use pre-trained (Word2Vec, GloVe, BERT)

## Reinforcement Learning:

- Agent learns from environment interaction
- Q-learning: value-based, tabular or deep (DQN)
- Applications: trading, portfolio optimization

**Key Takeaway:** Different tools for different problems

---

Course complete! Apply these methods in your capstone project

# References

## Embeddings:

- Mikolov et al. (2013). Word2Vec
- Pennington et al. (2014). GloVe
- Devlin et al. (2019). BERT

## Reinforcement Learning:

- Sutton & Barto (2018). RL: An Introduction (free online)
- Mnih et al. (2015). DQN (Atari)
- Schulman et al. (2017). PPO

## Finance Applications:

- Liu et al. (2021). FinRL: Deep RL for Trading
- Araci (2019). FinBERT

---

Sutton & Barto: the definitive RL textbook (free at [incompleteideas.net](http://incompleteideas.net))