L06: Embeddings & RL

Deep Dive: Theory, Implementation, and Applications

Methods and Algorithms

MSc Data Science

Spring 2026

Outline

Pouring Data into Linear Algebra

XKCD #1838 "Machine Learning" by Randall Munroe (CC BY-NC 2.5)

Learning Objectives

After this lecture, you will be able to:

1. **Derive** the Skip-Gram objective and negative sampling approximation
2. **Evaluate** static vs contextual embeddings (Word2Vec, GloVe, FinBERT)
3. **Analyze** Q-learning convergence and the exploration-exploitation trade-off
4. **Critique** RL trading strategies (transaction costs, non-stationarity, overfitting)

Finance Applications: Sentiment analysis with embeddings, algorithmic trading with RL

Bloom's Levels 4–5: Analyze, Evaluate, Derive, Critique

Part 1: Word Embeddings Introduction

The Problem with One-Hot Encoding

- Vocabulary of 10,000 words $\rightarrow$ 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 $\rightarrow$ 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

Skip-gram: Computational Challenge

Problem: The softmax denominator sums over **entire vocabulary**:

$$\sum_{w \in V} \exp(v_w^T v_{target}) \quad \text{— } O(|V|) \text{ per update!}$$

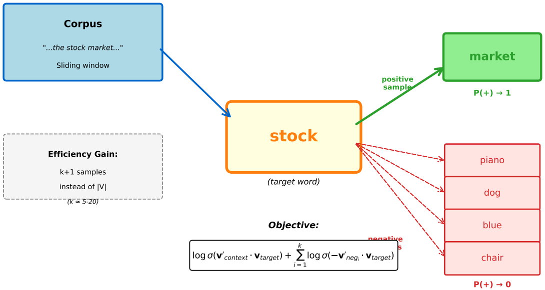
For $|V| = 100{,}000$ words, this is computationally intractable.

Solution: Negative Sampling (Mikolov et al., 2013b)
Replace full softmax with binary classification:

$$\log \sigma({v'_{w_O}}^T v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n} \left[\log \sigma(-{v'_{w_i}}^T v_{w_I}) \right]$$

- Positive pair: (target, true context) $\rightarrow$ predict 1
- k negative pairs: (target, random word) $\rightarrow$ predict 0
- Reduces $O(|V|)$ to $O(k)$ where $k = 5$–20

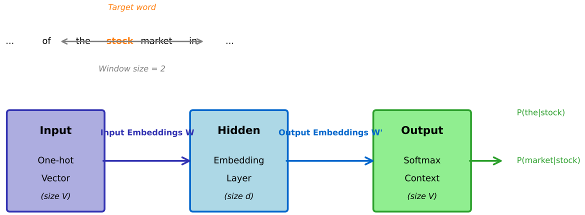
Negative sampling: the key innovation that made Word2Vec practical



Negative Sampling: Binary Classification Instead of Full Softmax

Binary classification: distinguish true context words from random “noise” words

Skip-gram Architecture



Skip-gram Architecture: Predict Context from Target

Two embedding matrices: input W (word vectors) and output W' (context vectors)

Skip-Gram with Negative Sampling: Algorithm

Require: corpus, embedding dim d, negatives k, window size, epochs

Initialize $W, W' \in \mathbb{R}^{|V| \times d}$ randomly
for each epoch **do**
  for each word w_t in corpus **do**
    for each context word w_c within window **do**
      Positive: update (w_t, w_c) to increase $\sigma(v_{w_t}^\top v'_{w_c})$
      for $i = 1, \ldots, k$ **do**
        Sample $w_n \sim P_n(w) \propto f(w)^{3/4}$
        Negative: update (w_t, w_n) to decrease $\sigma(v_{w_t}^\top v'_{w_n})$
      end for
    end for
  end for
end for
return W (word embeddings)

Key: Negative sampling (3-5 negatives per positive) replaces expensive softmax over entire vocabulary.

Mikolov et al. (2013). Distributed representations of words and phrases. NeurIPS, 3111–3119.

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

Word Analogy Limitations

Known Issues:

- Success rates typically 40–70%, not near 100% (Levy & Goldberg, 2014)
- Evaluation methodology inflates accuracy (nearest-neighbor dominance)
- Finance-domain analogies ($\vec{stock} - \vec{equity} + \vec{debt} \approx \vec{bond}$) not empirically validated

Bias in Embeddings:

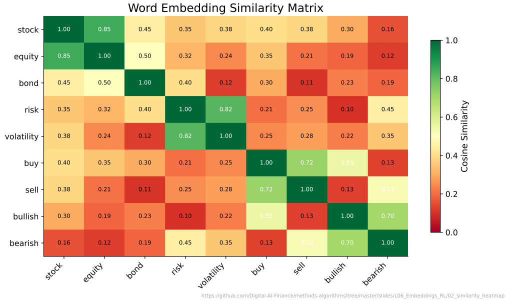
- Embeddings encode societal biases from training data (Bolukbasi et al., 2016)
- Example: man:programmer :: woman:homemaker
- **Finance concern**: Biased embeddings in credit scoring or hiring tools

Critical thinking: embeddings capture statistical patterns, including harmful ones

Cosine Similarity:

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

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



Cosine similarity ignores magnitude, focuses on direction

Pre-trained Embeddings

Popular Options:

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

Domain-Specific:

- **FinBERT**: BERT further pre-trained on financial corpora (Araci, 2019)
- BioBERT: Biomedical domain

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

Static vs Contextual Embeddings

Static Embeddings (Word2Vec, GloVe, FastText):

- ONE fixed vector per word, regardless of context
- "bank" in "river bank" = "bank" in "bank account"
- Fast, simple, good baseline

Contextual Embeddings (BERT, GPT, FinBERT):

- DIFFERENT vector per occurrence based on surrounding context
- "bank" gets different representations in different sentences
- State-of-the-art for most NLP tasks

Key Insight: Contextual models solve polysemy (multiple word senses)

Static: one meaning per word. Contextual: meaning depends on context.

Embeddings in Finance

Applications:

- **Sentiment Analysis**: News $\rightarrow$ embedding $\rightarrow$ 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 but loses word order: "bank robber" = "robber bank")
- Doc2Vec (paragraph vectors)
- Sentence-BERT (state-of-the-art)

Aggregate word embeddings to represent documents

Finance Example: Embedding-Based Sentiment

Task: Classify "Fed signals rate hike" as positive or negative

Step 1: Average word embeddings (simplified 3-dim vectors):

$$\vec{v}_{\text{sentence}} = \frac{1}{4}(\vec{v}_{\text{Fed}} + \vec{v}_{\text{signals}} + \vec{v}_{\text{rate}} + \vec{v}_{\text{hike}}) = [0.12, -0.31, 0.45]$$

Step 2: Compare to sentiment anchors via cosine similarity:

- $\text{sim}(\vec{v}_{\text{sentence}}, \vec{v}_{\text{positive}}) = 0.23$
- $\text{sim}(\vec{v}_{\text{sentence}}, \vec{v}_{\text{negative}}) = 0.61$

Step 3: Classify: **Negative sentiment** (rate hikes $\rightarrow$ tighter policy)

Real-world: Use FinBERT for production sentiment (up to 87% accuracy on financial text; Araci, 2019)

Simplified example — real embeddings are 300-768 dimensions with learned sentiment structure

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

The RL Loop:

Agent observes State $\rightarrow$ Agent selects Action $\rightarrow$ Environment transitions $\rightarrow$ Environment emits Reward $\rightarrow$ Agent updates $\rightarrow$ (repeat)

RL: Learning from interaction, not from labeled examples

Markov Decision Process

MDP Tuple: (S, A, P, R, γ)

- 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 (or $\gamma \in [0,1]$ for episodic tasks)

Markov Property:

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

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 π^*

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

TD(0) Update Rule — learn from each transition:

$$V(s) \leftarrow V(s) + \alpha \left[r + \gamma V(s') - V(s) \right]$$

TD Error: $\delta_t = r + \gamma V(s') - V(s)$ (surprise signal)

- **vs Monte Carlo**: MC waits for episode end; TD updates every step
- **vs Dynamic Programming**: DP requires model $P(s'|s, a)$; TD is model-free
- **Q-learning**: TD applied to Q-function with max over actions

TD learning: the theoretical foundation connecting DP, MC, and Q-learning (Sutton, 1988)

Q-Learning Algorithm

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 (ϵ-greedy)
   - Execute a, observe r, s'
   - Update $Q(s,a)$

Q-learning is off-policy: converges to Q^* given Robbins-Monro conditions ($\sum \alpha_t = \infty$, $\sum \alpha_t^2 < \infty$) and sufficient exploration (Watkins & Dayan, 1992)

Q-Learning: Worked Example

Trading scenario: State s_1 = [RSI=25, position=none]

Current Q-values: $Q(s_1, \text{buy}) = 3.2$, $Q(s_1, \text{hold}) = 1.0$

Agent takes action **buy**, observes:

- Reward $r = -0.5$ (transaction cost)
- New state s_2 = [RSI=35, position=long]
- Best future: $\max_{a'} Q(s_2, a') = 4.0$

Update ($\alpha = 0.1$, $\gamma = 0.9$):

$$\underbrace{r + \gamma \max_{a'} Q(s_2, a')}_{\text{TD target}} - \underbrace{Q(s_1, \text{buy})}_{\text{current}} = -0.5 + 0.9 \times 4.0 - 3.2 = -0.1$$

$$Q(s_1, \text{buy}) \leftarrow 3.2 + 0.1 \times (-0.1) = \mathbf{3.19}$$

Each update moves Q toward the "better" estimate: immediate reward + discounted future

Q-Learning Algorithm: Pseudocode

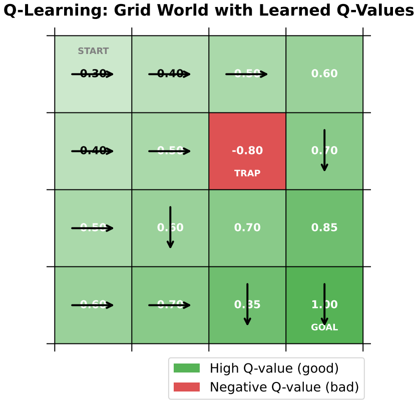
Require: environment, α, γ, ϵ, episodes

Initialize $Q(s,a) \leftarrow 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
for episode $= 1, \ldots,$ episodes **do**
  $s \leftarrow$ initial state
  while s is not terminal **do**
    $a \leftarrow \begin{cases} \text{random } a \in \mathcal{A} & \text{with prob. } \epsilon \\ \arg\max_{a'} Q(s,a') & \text{otherwise} \end{cases}$ {ϵ-greedy}
    Take action a, observe reward r and next state s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$
  end while
end for
return Q

Key: The $\max_{a'}$ makes Q-learning **off-policy** — it learns the optimal policy regardless of the exploration strategy used.

Watkins & Dayan (1992). Q-learning. Machine Learning, 8(3-4), 279–292.

Q-Values Visualization



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)

ϵ-**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 ϵ (explore more)
- Decay ϵ over time (exploit more)

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

RL for Trading

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

Trading Reward Function Design

Reward with transaction costs:

$$r_t = R_t^{\text{portfolio}} - c \cdot |\Delta w_t|$$

where $R_t^{\text{portfolio}}$ = portfolio return, c = transaction cost rate, Δw_t = position change

Common State Features:

- Price returns (1-day, 5-day, 20-day)
- Technical indicators: RSI, MACD, Bollinger width
- Current position and unrealized P&L

Alternative Rewards:

- Sharpe ratio: $r_t = \frac{\bar{R}_t}{\sigma_{R_t}}$ (risk-adjusted, but non-stationary)
- Log return: $r_t = \log(1 + R_t)$ (additive over time)

Reward design is THE most critical decision in RL for trading

Backtesting RL Trading Strategies

Critical Challenge: RL agents overfit to historical data

Walk-Forward Validation:

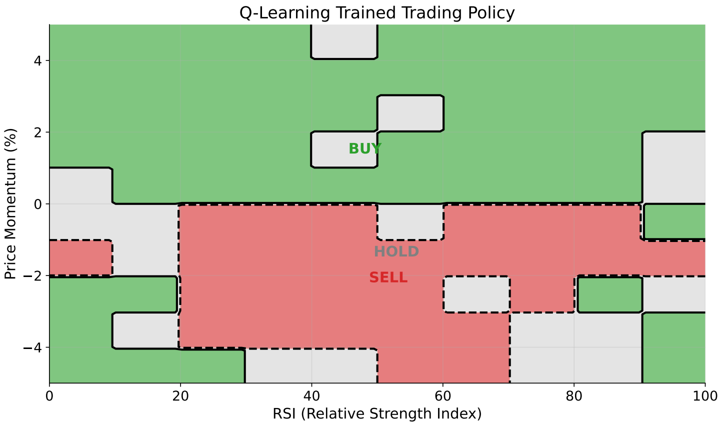
1. Train on period $[t_0, t_1]$, test on $[t_1, t_2]$
2. Roll forward: train on $[t_1, t_2]$, test on $[t_2, t_3]$
3. Report average out-of-sample performance

Honest Evaluation:

- Compare to buy-and-hold benchmark (most RL strategies fail to beat after costs)
- Include realistic transaction costs (0.1–0.5% per trade)
- Test across multiple market regimes (bull, bear, sideways)

If your RL agent beats buy-and-hold after costs, you likely have a bug — verify carefully

Trading Policy



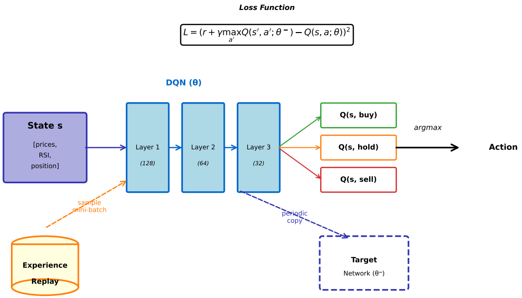
https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L06_Embeddings_RL/06_policy_viz

Q-learning trained policy: agent discovers buy/sell/hold regions from reward signal

Idea: Neural network approximates Q-function: $Q(s, a; \theta) \approx Q^*(s, a)$
Loss Function:

$$L(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

Key Innovations:
- **Experience Replay**: Store (s, a, r, s'), sample random mini-batches (breaks temporal correlation)
- **Target Network** θ^-: Separate, slowly-updated copy for stability



Deep Q-Network Architecture for Trading

DQN: Atari-level play from raw pixels (Mnih et al., 2015); loss is mean squared TD error

Policy Gradient Methods

Policy Gradient Theorem (Sutton et al., 2000):

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

where $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ is the **advantage function**

- **REINFORCE**: Uses episode returns G_t as A; high variance
- **Actor-Critic**: Actor (policy π_θ) + Critic (learns V^ϕ); lower variance
- **PPO**: Clips policy ratio to prevent large updates; widely used

Policy gradient handles continuous actions; advantage reduces variance vs raw returns

Embedding Uncertainty:

- Bootstrap cosine similarity: resample corpus, retrain, compute CI
- Permutation test: shuffle word-context pairs, check if similarity is significant

RL Uncertainty:

- Q-value confidence: run N independent training runs, report mean $\pm$ std
- Off-policy evaluation: importance sampling to estimate policy value from logged data

$$\hat{V}(\pi) = \frac{1}{n} \sum_{i=1}^{n} \prod_{t=0}^{T} \frac{\pi(a_t|s_t)}{\beta(a_t|s_t)} \cdot G_i$$

Always report uncertainty — a single training run is not evidence of a good policy

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/github/Digital-AI-Finance/methods-algorithms/blob/master/notebooks/L06_embeddings_rl.ipynb`

When to Use What

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

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

Practical Tips

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

Key Takeaways

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

Closing Thought

"After six lectures of methods and algorithms,
we've learned the most important lesson:

pour the data into the right pile of linear algebra,
and the answers will come out the other side.

The hard part is knowing which pile.'"

— Adapted from XKCD #1838 "Machine Learning" by Randall Munroe

References: Embeddings

- Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *NeurIPS*, 3111–3119.
- Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global vectors for word representation. *EMNLP*, 1532–1543.
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers. *NAACL*, 4171–4186.
- Levy, O. & Goldberg, Y. (2014). Linguistic regularities in sparse and explicit word representations. *CoNLL*, 171–180.
- Bolukbasi, T., Chang, K.-W., Zou, J., Saligrama, V., & Kalai, A. (2016). Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. *NeurIPS*, 4349–4357.

References: RL and Finance

- Sutton, R. & Barto, A. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.
- Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529–533.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv:1707.06347*.
- Watkins, C. & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4), 279–292.
- Liu, X.-Y., Yang, H., Gao, J., & Wang, C. (2021). FinRL: Deep reinforcement learning framework for automated trading. *SSRN*.
- Araci, D. (2019). FinBERT: Financial sentiment analysis with pre-trained language models. *arXiv:1908.10063*.

Sutton & Barto: the definitive RL textbook (free at incompleteideas.net)

Appendix

Advanced Topics and Proofs

Supplementary material for self-study and reference

Appendix slides are not covered in lecture — provided for advanced students and exam preparation.

Skip-Gram Objective Derivation

Maximum Likelihood Objective:
Given corpus of word-context pairs (w_t, w_c), maximize:

$$\mathcal{L} = \sum_{(w_t, w_c) \in D} \log P(w_c | w_t)$$

With softmax parameterization:

$$\log P(w_c | w_t) = v_{w_c}'^{\top} v_{w_t} - \log \sum_{w \in V} \exp(v_w'^{\top} v_{w_t})$$

Simplification: The log-sum-exp term is the log-partition function. Maximizing $\mathcal{L}$ is equivalent to minimizing cross-entropy between the model distribution and the empirical context distribution.

Connection to cross-entropy:

$$H(p_{\text{empirical}}, p_{\text{model}}) = -\sum_{w_c} \hat{p}(w_c | w_t) \log p_\theta(w_c | w_t)$$

Skip-Gram is a discriminative model: it models $P(\text{context}|\text{target})$ directly, not a generative process

Negative Sampling Theory

Origin: Noise Contrastive Estimation (NCE)

- NCE (Gutmann & Hyvärinen, 2012): estimate unnormalized models by contrasting data with noise
- Negative sampling is a simplified variant of NCE

Why the $3/4$ Power?

- Noise distribution: $P_n(w) \propto f(w)^{3/4}$ where $f(w)$ is unigram frequency
- Exponent < 1 upweights rare words relative to frequency
- Empirically chosen by Mikolov et al. (2013) — not theoretically derived

Implicit Matrix Factorization (Levy & Goldberg, 2014):
SGNS implicitly factorizes a shifted PMI matrix:

$$v_w \cdot v'_c \approx \mathrm{PMI}(w, c) - \log k$$

where $\mathrm{PMI}(w, c) = \log \frac{P(w,c)}{P(w)P(c)}$ and k = number of negatives

Negative sampling implicitly factorizes a shifted PMI matrix

Bellman Equation Convergence

Robbins-Monro Conditions for step sizes α_t:

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

Contraction Mapping Argument:

- The Bellman optimality operator T is a γ-contraction in ℓ_∞-norm
- $\|TQ_1 - TQ_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$
- By Banach fixed-point theorem, T has unique fixed point Q^*
- Q-learning converges to Q^* when Robbins-Monro conditions hold and all (s, a) pairs visited infinitely often

Fixed α Issue:

- Constant α violates $\sum \alpha_t^2 < \infty$ — Q-values oscillate around Q^*
- In practice: fixed α works well in non-stationary environments (tracks changes)

Watkins & Dayan (1992): formal convergence proof requires decaying step sizes and full exploration

DQN Architecture Details

Experience Replay Buffer:

- Store transitions $(s, a, r, s', \text{done})$ in buffer of size N (e.g., 10^6)
- Sample uniform random mini-batches for training
- Breaks temporal correlation $\rightarrow$ approximately i.i.d. data

Target Network Updates:

- **Hard update**: Copy $\theta^- \leftarrow \theta$ every C steps (Mnih et al., 2015)
- **Soft update**: $\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$ with $\tau \ll 1$ (Polyak averaging)

Extensions:

- **Double DQN** (van Hasselt et al., 2016): Decouple action selection from evaluation to reduce overestimation bias
- **Dueling DQN** (Wang et al., 2016): Separate value $V(s)$ and advantage $A(s, a)$ streams: $Q(s, a) = V(s) + A(s, a) - \bar{A}(s)$

Mnih et al. (2015): DQN achieved human-level play on 29/49 Atari games

Word Embedding Bias and Fairness

Bias in Word Embeddings (Bolukbasi et al., 2016):

- Gender: he:doctor :: she:nurse (reflects training corpus stereotypes)
- Race, religion, and other protected attributes similarly affected

Debiasing Techniques:

- **Post-hoc projection**: Remove gender direction from embedding space
- **Counterfactual data augmentation**: Balance training examples
- **Adversarial debiasing**: Train to be invariant to protected attributes

Finance Implications:

- Biased embeddings in **credit scoring** can violate fair lending laws
- **Hiring tools** using biased embeddings risk discrimination claims
- **EU AI Act**: High-risk AI systems (credit, hiring) require bias auditing

Embedding bias is a compliance risk in regulated financial services

SARSA vs Q-Learning

SARSA (on-policy):

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

Uses the **actual next action** a' chosen by the current policy.

Q-Learning (off-policy):

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

Uses the **greedy maximum** regardless of what action was actually taken.

Key Differences:

- **Safety**: SARSA accounts for exploration risk; Q-learning ignores it
- **Cliff-walking example**: SARSA learns the safe path (away from cliff edge); Q-learning learns the optimal but risky path (along the edge)
- **Convergence**: Both converge given Robbins-Monro conditions; Q-learning to Q^*, SARSA to Q^π

SARSA: safer path; Q-learning: optimal path

Appendix References and Further Reading

- Levy, O. & Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. *NeurIPS*, 2177–2185.
- Bolukbasi, T. et al. (2016). Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. *NeurIPS*, 4349–4357.
- Watkins, C. & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4), 279–292.
- van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. *AAAI*, 2094–2100.
- Wang, Z., Schaul, T., Hessel, M., et al. (2016). Dueling network architectures for deep reinforcement learning. *ICML*, 1995–2003.
- Sutton, R. & Barto, A. (2018). *Reinforcement Learning: An Introduction*, Chapters 6 and 16. MIT Press.

All appendix references are freely available online