Final Report: Policy Optimization for Financial Decision-Making

Author: Nayan Jain **Date:** October 26, 2025

---

1. Executive Summary

This project sought to develop an intelligent loan approval system to maximize financial returns for a fintech company. We compared two distinct machine learning paradigms: a traditional **Supervised Deep Learning (DL)** model trained to *predict risk* and an **Offline Reinforcement Learning (RL)** agent trained to *optimize a policy* for maximum profit.

The key finding is that while the DL model proved effective at identifying risk (AUC: 0.716), its objective is only a proxy for the true business goal. The RL agent, trained on the same data, learned a policy with an **Estimated Policy Value of $962.9542** per loan. This represents a potential **improvement of over $2335.20** per loan compared to the bank's historical policy (which averaged a loss of -$1372.25 per loan). This result strongly indicates that RL is a superior framework for this business problem, as it optimizes for the financial outcome directly.

---

2. Project Methodology

2.1. Dataset and Preprocessing

We used the LendingClub Loan Data (https://www.kaggle.com/datasets/wordsforthewise/lending-club) (`accepted_2007_to_2018.csv`).

- **Target & Reward Engineering:** We filtered the data to loans with a final status.
  - **DL Target:** `is_default` (0 for "Fully Paid", 1 for "Charged Off" or "Default").
  - **RL Reward:** A financial reward was engineered:
    - `Action = Deny`: Reward = $0
    - `Action = Approve & Fully Paid`: Reward = `loan_amnt * int_rate` (Profit)
    - `Action = Approve & Defaulted`: Reward = `-loan_amnt` (Loss)
- **Feature Selection:** We selected 13 features representing the loan conditions, the borrower's credit history, and their financial capacity (e.g., `loan_amnt`, `int_rate`, `fico_range_low`, `dti`, `annual_inc`).
- **Preprocessing:** A full `sklearn` pipeline was built to handle missing values, one-hot encode categorical features, and apply `StandardScaler` to all numeric features.

2.2. Model 1: The Predictive Deep Learning (DL) Model

- **Architecture:** A Multi-Layer Perceptron (MLP) in PyTorch with Batch Normalization and Dropout.
- **Objective:** To predict the probability of `is_default`.
- **Training:** A robust training harness was used, including a validation set, early stopping, and a learning rate scheduler to prevent overfitting and find the best model.
- **Evaluation:** The model was evaluated on its ability to classify risk on a held-out test set.

2.3. Model 2: The Offline Reinforcement Learning (RL) Agent

- **Framework:** We framed the problem as an offline RL task.
  - **State (s):** The preprocessed feature vector of the applicant.
  - **Action (a):** A discrete space {0: Deny, 1: Approve}.
  - **Reward (r):** The engineered financial reward.
- **Algorithm:** We used **Discrete-CQL (Conservative Q-Learning)**. This algorithm is ideal for our problem as it's designed for offline datasets (learning from past data only) and prevents the agent from overestimating the value of actions (like "Deny") that it never saw in the historical data.
- **Evaluation:** We used **DiscreteFQE (Fitted Q Evaluation)** to train a separate Q-model to estimate the expected financial return (the "Policy Value") our new RL policy would achieve.

---

3. Results and Analysis

3.1. Key Performance Metrics

Model	Metric	Value
Model 1 (DL)	AUC (Area Under ROC Curve)	**0.716**
Model 1 (DL)	F1-Score (tuned threshold)	**0.432**
Historical Policy	Actual Average Reward	**-$1372.25**
Model 2 (RL)	Estimated Policy Value	**$962.9545**

3.2. Explaining the Difference in Metrics

A crucial part of this project is understanding *why* we use different metrics for each model.

- **Why AUC and F1-Score for the DL Model?**
  - **AUC (0.716)** tells us the model is good at *ranking* applicants. It's significantly better than a random guess (0.5) at distinguishing between a "good" loan and a "bad" loan, regardless of any specific threshold.
  - **F1-Score (0.43)** is the harmonic mean of precision and recall. For an imbalanced dataset, it's a better measure of classification accuracy than "accuracy" alone. It shows the model's ability to find "Defaulted" loans (recall) without incorrectly flagging too many "Fully Paid" loans (precision).
  - **Conclusion:** These are *proxy* metrics. They measure *predictive accuracy*, not *business outcome*.

- **Why "Estimated Policy Value" for the RL Agent?**
    - **Policy Value ($962.9545)** is the *direct business metric*. It answers the question, "If we use this agent's approval policy, how much money (in dollars) will we make or lose on average for every loan application we process?"
    - This metric moves beyond simple prediction. It represents the *expected financial return* of the learned decision-making policy. This is precisely what the business wants to maximize. The **$2335.20** improvement over the historical loss of -$1372.25 quantifies the business value of our new model.

3.3. Comparing the Policies: The "Profit Optimizer" vs. The "Risk Avoider"

The DL and RL models define two very different types of policies.

- The **DL model** creates an **implicit policy**. It only provides a risk score (e.g., 65% chance of default). A human or business rule must then set a threshold (e.g., "deny all loans with >40% risk"). This policy is a pure "risk avoider."

- The **RL agent** learns an **explicit policy**. It directly outputs or for any given applicant. This policy is a "profit optimizer."

Case Study: The High-Risk, High-Interest Applicant

Let's consider an applicant who is high-risk but has been offered a very high interest rate.

- **The DL Model** would flag this applicant with a high probability of default. If our threshold is 40%, and the model predicts 50% risk, it would recommend **DENY**. It *only* sees the risk.
- **The RL Agent** would see both the risk *and* the reward. It would internally calculate:
    - *Potential Loss:* `-loan_amnt` (a high-risk event)
    - *Potential Profit:* `loan_amnt * high_int_rate` (a high-reward event)
    - If the potential profit outweighs the weighted risk, the RL agent will recommend **APPROVE**. It understands that taking a calculated risk is necessary for profit.

The RL agent learns a more nuanced policy that the bank *should* approve certain risky loans if the financial upside is high enough, a decision a simple risk-prediction model would never make.

---

4. Limitations and Future Steps

While the RL agent shows immense promise, this project has limitations inherent to any offline data problem.

4.1. Limitations

1. **Counterfactuals (What Ifs?):** Our dataset only contains *approved* loans. We have no data on what would have happened if the bank had approved the loans it *rejected*. The agent is learning from a heavily biased sample.

2. **Simple Reward Model:** Our reward function (`-loan_amnt`) is simple. In reality, a defaulted loan has recovery costs, legal fees, and potential to recover *some* of the principal, making the true loss variable.
3. **Static States:** We only look at the applicant's data at the *time of application*. We don't consider how their financial situation might change, which could be a source of data for a more advanced model.

4.2. Future Steps

Based on these findings, I would not recommend immediate, full-scale deployment. I would propose the following next steps:

1. **A/B Testing (Online Evaluation):** The most critical next step. Deploy the RL agent's policy in a "shadow mode" (making decisions but not acting on them) or in a limited A/B test (e.g., giving it 5% of new applications). This is the only way to safely and accurately measure its true real-world performance.
2. **Explore GBDTs for Prediction:** For tabular data, Gradient-Boosted Decision Trees (like XGBoost or LightGBM) often outperform MLPs for the *prediction* task. This could serve as an even stronger baseline.
3. **Refine the Reward Model:** Work with the finance department to build a more accurate reward function that includes costs of collection and partial recoveries.
4. **Collect Better Data:** If possible, begin a small pilot program to approve a random selection of "borderline" rejected loans to gather data on their outcomes. This would provide the model with crucial, unbiased data to learn from.