APPLIED DEEP LEARNING

NEURAL NETWORK CLASSIFIER FOR LOAN APPROVAL

BY: SALWA NIAZ PREET

Contents

Problem Statement

The objective of this study is to identify the key determinants of loan approval and to build a robust neural-network–based model for predicting whether an applicant will be approved or denied. I will analyze how borrower demographics (age, gender, education), financial profile (income, work experience, home-ownership), credit history (bureau history length, credit score, prior defaults), and loan terms (amount requested, intended use, interest rate, debt-to-income ratio) jointly affect the probability of approval. By training a binary-classification network on these features, I aim to deliver a decision-support tool that helps lenders make informed underwriting decisions and assists applicants in understanding their likelihood of securing a loan.

Dataset

Each row in the dataset corresponds to an individual application. The dataset contains a rich set of borrower demographics, financials, credit-history and loan-specific information used to predict whether the loan was approved or not. Key fields include:

- **Demographics:** person_age, person_gender, person_education (Associate, High School, Bachelor, Master, Doctorate)
- **Financial profile:** person_income, person_emp_exp (years of employment), person_home_ownership (Rent, Own, Mortgage, Other)
- **Loan details:** loan_amnt, loan_intent (Personal, Education, Medical, Venture, Homeimprovement, Debtconsolidation), loan_int_rate, loan_percent_income (loan amount as a fraction of income)
- **Credit history:** cb_person_cred_hist_length (years of credit history), credit_score, previous_loan_defaults_on_file (yes, no)
- **Outcome:** loan_status (1 = approved, 0 = not approved)

Together, these 14 variables will help train the binary-classification model that estimates each applicant's approval based on their background, creditworthiness, and the terms of the loan.

Data Preprocessing

The data set was divided into categorical and numerical features where categorical included person gender, person education, person homeownership, loan intent, previous loan defaults on file. Of these, person education and previous loan defaults on file were label encoded and all others were one-hot-encoded.

Neural network architecture and hyperparameters

- **Model Structure**
    - **Input layer:** accepts the full vector of scaled features (X_train_scaled.shape[1]).
    - **Hidden layers (2):**

        - **Layer 1:** Dense with tunable units (32 - 128, step 32) + Dropout (rate 0.1 - 0.5, step 0.1)
        - **Layer 2:** Dense with tunable units (32 - 128, step 32) + same Dropout rate as Layer 1
    - **Output layer:** single neuron with sigmoid activation for binary classification.
- **Activation Functions**
    - Choice among ReLU, tanh, or sigmoid, selected globally for both hidden layers.
- **Regularization**
    - **Dropout:** rate tuned between 0.1 and 0.5 to prevent overfitting.
    - **Early Stopping:** monitors validation loss, with patience=3, minimum delta=0.001, and restore_best_weights=True during training.
- **Optimizers & Learning Rates**
    - **Optimizer choice:** Adam, Adagrad, or RMSprop
    - **Learning rate:** one of [1e-2, 1e-3, 1e-4]
    - All models compiled with loss=binary cross entropy and metrics=accuracy
- **Hyperparameter Tuning Setup**
    - **Methods used:**
        - Random Search (max trials=10, max epochs=3)
        - Bayesian Optimization (max trials=10, max epochs=3)
        - Hyperband (max epochs=3)
    - Each tuner runs with the same search budget and uses the Early Stopping callback on validation loss.
- **Evaluation Threshold**
    - Final classification threshold set at 0.50 on predicted probability

Results

Method	Test Loss	Time (seconds)
Random Search	0.214048	185.682327
Bayesian Optimization	0.204749	167.030122
Hyperband	0.209148	64.434670

Bayesian Optimization achieved the lowest test loss (0.2047) at a moderate runtime, indicating the best-quality hyperparameters. Hyperband was the fastest method (≈64s) but with a slightly higher loss (0.2091), while Random Search was the slowest (≈186 s) and produced the highest loss (0.2140).

Hyperband's adaptive "successive halving" strategy lets you explore many configurations very quickly by early-stopping poorly performing trials. In this case it reached a competitive test loss (0.2091) in just ~64 seconds, about one-third the time of Random Search or Bayesian Optimization, making it ideal whenever we need fast, reliable hyperparameter tuning under tight compute or time constraints.

Best hyperparameters

Random Search

- 1st hidden layer: 64 units, sigmoid activation
- Dropout rate: 0.40
- 2nd hidden layer: 96 units
- Optimizer: RMSprop with learning rate 0.01

Bayesian Optimization

- 1st hidden layer: 64 units, ReLU activation
- Dropout rate: 0.40
- 2nd hidden layer: 64 units
- Optimizer: Adam with learning rate 0.01

Hyperband

- 1st hidden layer: 96 units, ReLU activation
- Dropout rate: 0.30
- 2nd hidden layer: 32 units
- Optimizer: RMSprop with learning rate 0.001

Test Accuracy: 0.9063

Confusion Matrix

Actual \ Predicted	0 (No Approval)	1 (Approval)
0 (No Approval)	6615	392
1 (Approval)	451	1542

True Positive: 1542
True Negative: 6615
False Positive: 392
False Negative: 451

Classification Report & Explanation

Class	Precision	Recall	F1-Score	Support
0	0.94	0.94	0.94	7007
1	0.80	0.77	0.79	1993
Accuracy			0.9063	9000
Macro avg	0.87	0.86	0.86	9000
Weighted avg	0.91	0.91	0.91	9000

Class = 1 (Approved = Yes)

- **Precision = 0.80**

- Of all loans the model flagged as 'Approved,' 80% actually did get approved.
- In practice, this means 20% of the time the model wrongly labels an approved loan as not approved (false positive).

- **Recall = 0.77**
  - The model correctly identifies 77% of all actual approvals.
  - Equivalently, it misses 23% of true approvals (false negatives), letting some not approved applications slip through.
- **F1-Score = 0.79**
  - The harmonic mean of precision and recall balances both concerns.
  - A 0.79 F1 indicates the model maintains a reasonable trade-off between predicting approved applications and avoiding false alarms.

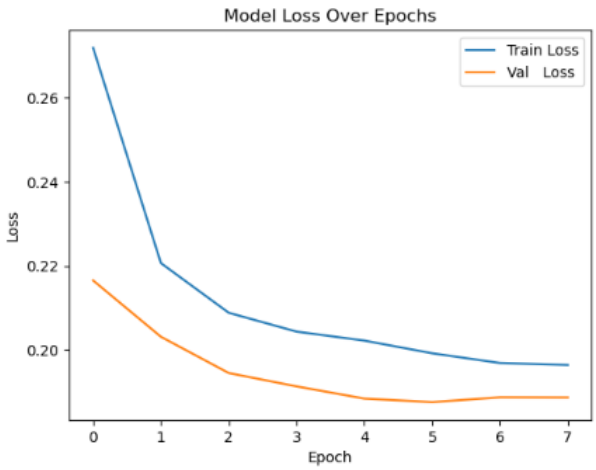
Class = 0 (Not Approved)

- **Precision = 0.94**
  - When the model predicts 'Not Approved,' it is correct 94% of the time
- **Recall = 0.94**
  - It recovers 94% of all actually non-approved borrowers, ensuring most good applicants are approved.

Overall, the model is more conservative on approving loans (high non-approved precision/recall) while achieving moderately strong performance on detecting approvals (0.80 precision, 0.77 recall).

Training & Validation Loss Over Epochs

The loss curves show that both training and validation losses drop quickly in the first few epochs and then level off around epoch 5-7. Because the validation loss tracks, and even dips slightly below, the training loss without diverging, it suggests the model is converging well without overfitting. The plateau after epoch 5 indicates that further training yields diminishing returns, precisely why I use early stopping around that point.



LIME Explainability

After training, I used LIME to open the “black box” for a single test loan and see exactly which inputs drove its approval prediction:

I invoked LimeTabularExplainer on the scaled training data and then called explain_instance for one test sample.

What it showed

Feature	Weight
0.00 < remainder__previous_loan_defaults_on_file <= 1.00	-0.370027
cat__loan_intent_VENTURE <= 0.00	0.109825
cat__person_home_ownership_OWN <= 0.00	0.107006
cat__loan_intent_PERSONAL > 0.00	-0.0580168
cat__loan_intent_EDUCATION <= 0.00	0.0566351
num__credit_score <= -0.62	0.05543
num__person_income <= -0.39	0.0502421
0.00 < cat__person_home_ownership_RENT <= 1.00	0.0469727
cat__loan_intent_HOMEIMPROVEMENT <= 0.00	0.0389378
cat__person_home_ownership_OTHER <= 0.00	0.0312571

- **Feature**
  Each row shows a threshold feature/rule. If the given one-hot-encoded feature is less than 0, that implies absence of the given category and if it is given as more than 0, it means presence of that specific category.
- **Weight**
  A positive weight pushes the prediction toward Approval (1); a negative weight pushes it toward No Approval (0). The magnitude tells how strong that push is for this particular loan.

Interpretations

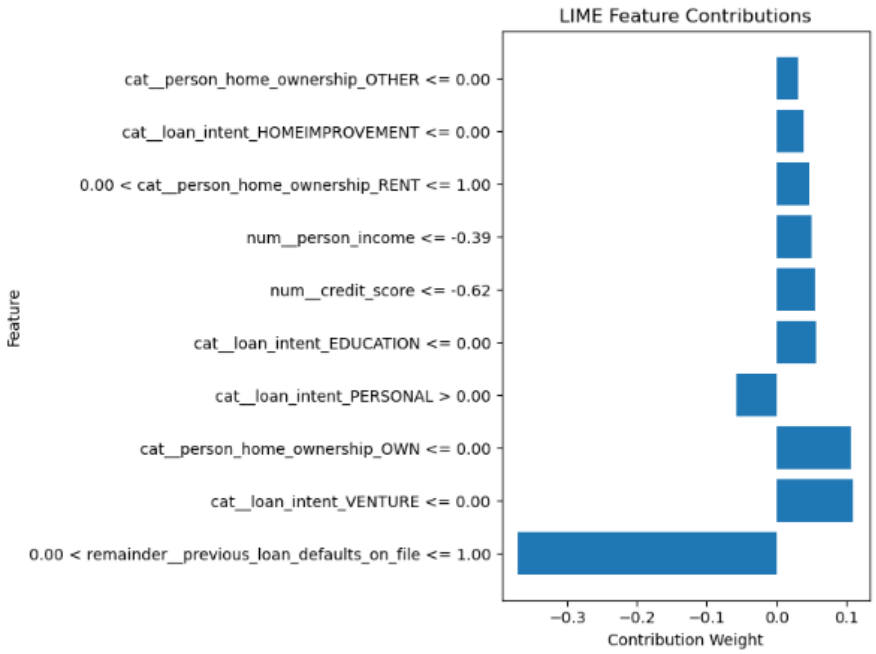
1. 0.00 < remainder__previous_loan_defaults_on_file <= 1.00 ⇒ −0.37
   - Having a previous loan default decreases approval probability by about 0.37.
2. cat__loan_intent_VENTURE <= 0.00 ⇒ +0.11
   - If loan is not for “Venture” purpose, it increases approval probability by 0.11.
3. cat__person_home_ownership_OWN <= 0.00 ⇒ +0.11
   - If applicant does not own their home, it gives a +0.11 boost toward approval.
4. cat__loan_intent_PERSONAL > 0.00 ⇒ −0.06
   - If the loan is “Personal”, it slightly lowers approval chances by 0.06.
5. cat__loan_intent_EDUCATION <= 0.00 ⇒ +0.06
   - If loan is not an “Education” loan, it increases approval chance by 0.06.
6. num__credit_score <= −0.62 ⇒ +0.06
   - A relatively high credit score nudges approval up by 0.06.
7. num__person_income <= −0.39 ⇒ +0.05

- A higher income (above that cutoff) adds 0.05 toward approval.

8. 0.00 < cat__person_home_ownership_RENT <= 1.00 ⇒ +0.05
    - If the borrower rents, there is a mild +0.05 approval boost.
9. cat__loan_intent_HOMEIMPROVEMENT <= 0.00 ⇒ +0.04
    - Not a “Home Improvement” loan, pushing approval by +0.04.
10. cat__person_home_ownership_OTHER <= 0.00 ⇒ +0.03
    - Not “Other” home ownership, a small +0.03 lift toward approval.

KEY TAKEAWAY:

- Largest negative driver (−0.37) is prior defaults, which most strongly hurts approval.
- Positive drivers (loan purpose ≠ Venture/Personal/Education and strong credit/income) each add modest approval probability.



Challenges & Solutions

- **Preventing Overfitting**
    - *Challenge:* Early experiments showed the network quickly fit training data but then validation loss plateaued or rose even more than training
    - *Solution:* I added tunable Dropout layers after each hidden Dense layer and an EarlyStopping callback (monitoring val_loss with patience=3, min_delta=0.001) to halt training once generalization diminished.
- **Balancing Tuning Speed and Accuracy**

    - *Challenge:* Random Search and Bayesian Optimization each took several minutes per run, slowing iteration.
    - *Solution:* I compared all three methods (Random Search, Bayesian, and Hyperband) and ultimately adopted Hyperband when under time constraints, since it reached near-optimal loss in roughly one-third the time.

- **Improving LIME Visualizations**
    - *Challenge:* LIME's default force-plot bars and rule text were truncated at the notebook margins.
    - *Solution:* I separated the explanation into two cells: first printing a pandas table of feature–weight pairs, then drawing a standalone horizontal bar chart with plt.tight_layout() to ensure full visibility of feature names.

- **Understanding LIME Outputs**
    - *Challenge:* LIME was new to me and its rule-based explanations (e.g. "feature ≤ threshold") took time to parse and validate.
    - *Solution:* Spent time researching the underlying methodology, reviewing tutorials, and experimenting with multiple instances, so I could correctly interpret feature contributions and ensure my explanations matched the encoded data.

- **Managing Package Dependencies**
    - *Challenge:* Occasional "ModuleNotFoundError" when importing LIME in different environments.
    - *Solution:* I standardized cells to begin with !pip install lime -q and then the appropriate import statements, ensuring reproducibility on local machines.

These solutions not only resolved immediate errors but also made the preprocessing, tuning, and explainability pipelines more robust and easier to maintain.