Deep Recurrent Neural Networks for Fraud Detection on Debit Card Transactions

Antonio Martini
Barclays, Quantitative Analytics, Fraud Detection

Introduction

- Current models for detecting fraud on debit card transactions rely on classical machine learning approaches.

- Recent advances in Deep Learning and Recurrent Neural Networks(RNN) in particular, offer the opportunity to make substantial improvements over more traditional approaches.

- We share our latest research results on applying RNN to fraud detection where we achieve an overall fraud detection rate of 35%. This is a substantial improvement over the 10% of the incumbent model and it looks promising when compared to the 40% achieved by Gradient Boosted Tree based 1st gen models.

- We discuss the required improvements that will likely allow RNNs to surpass GBT models in the near future.

Current Models

Classical ML models for fraud detection on card transactions.

Two model types:

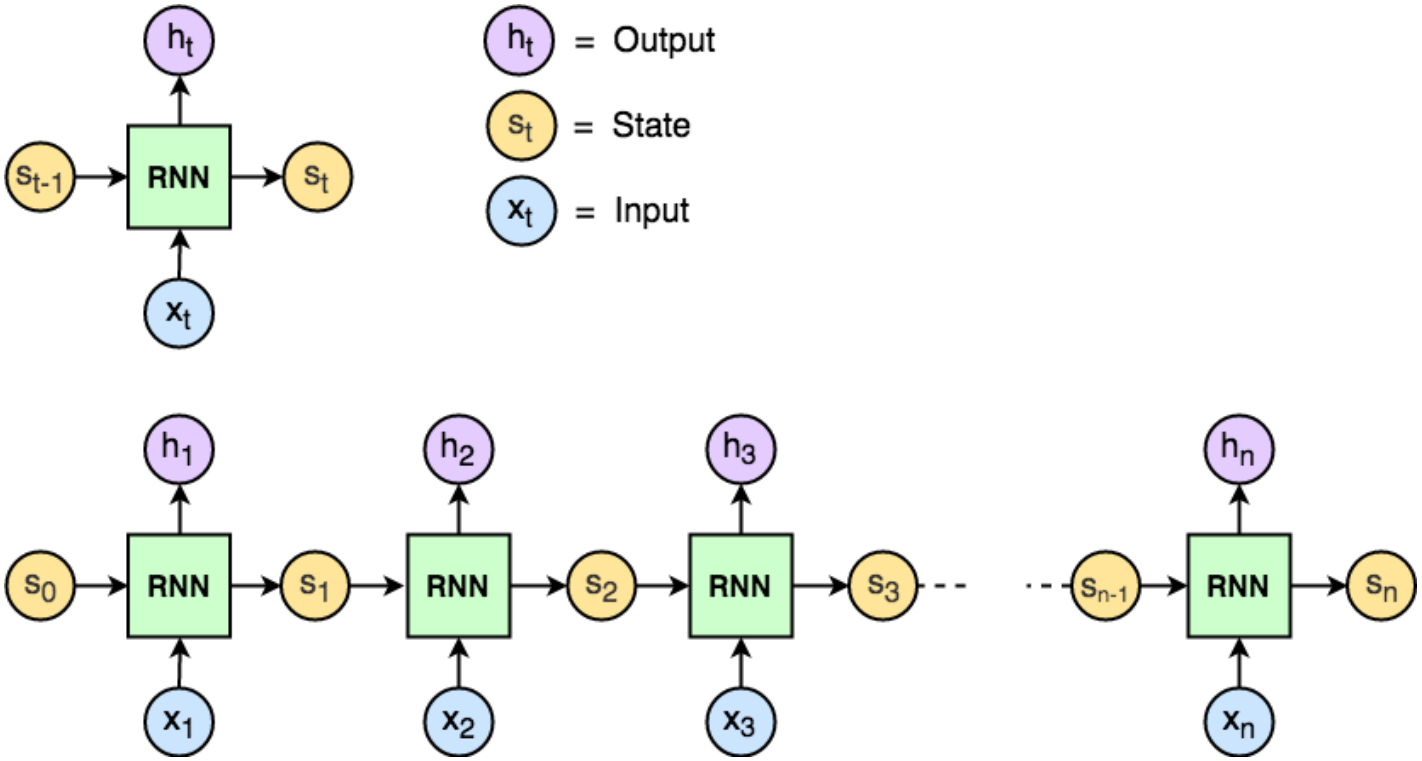
- Incumbent Model: Shallow Feedforward Networks.
- 1st Gen. Internal Model: Gradient Boosted Trees.

Properties:

- Two separate models: Card-Present(CP), Card-Not-Present(CNP).
- Temporal information is accounted for by features derived from aggregated statistics(e.g. mean, count, sum over a time window etc..)
- The form of the aggregated statistics is predefined (e.g. mean, sum, count) and so are the aggregation windows(1 hour, 1 day , 1 week etc..)
- Features creation is a long and tedious process.

Recurrent Neural Networks(RNN)

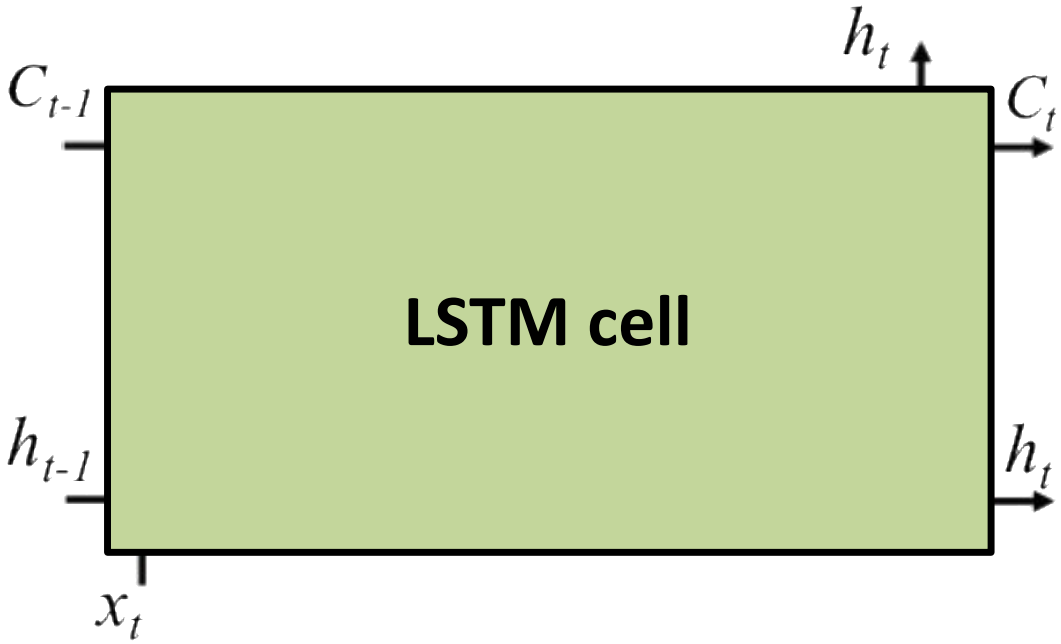
Naturally handle time-series data .The automatically computed RNN state efficiently summarizes the time-series past.



- Potentially improved predictions.
- Reduced feature engineering.
- A single model for CP/CNP transactions.

Long Short Term Memory (LSTM)

Employ a memory cell architecture that allows learning over long sequences.



x_t : Input

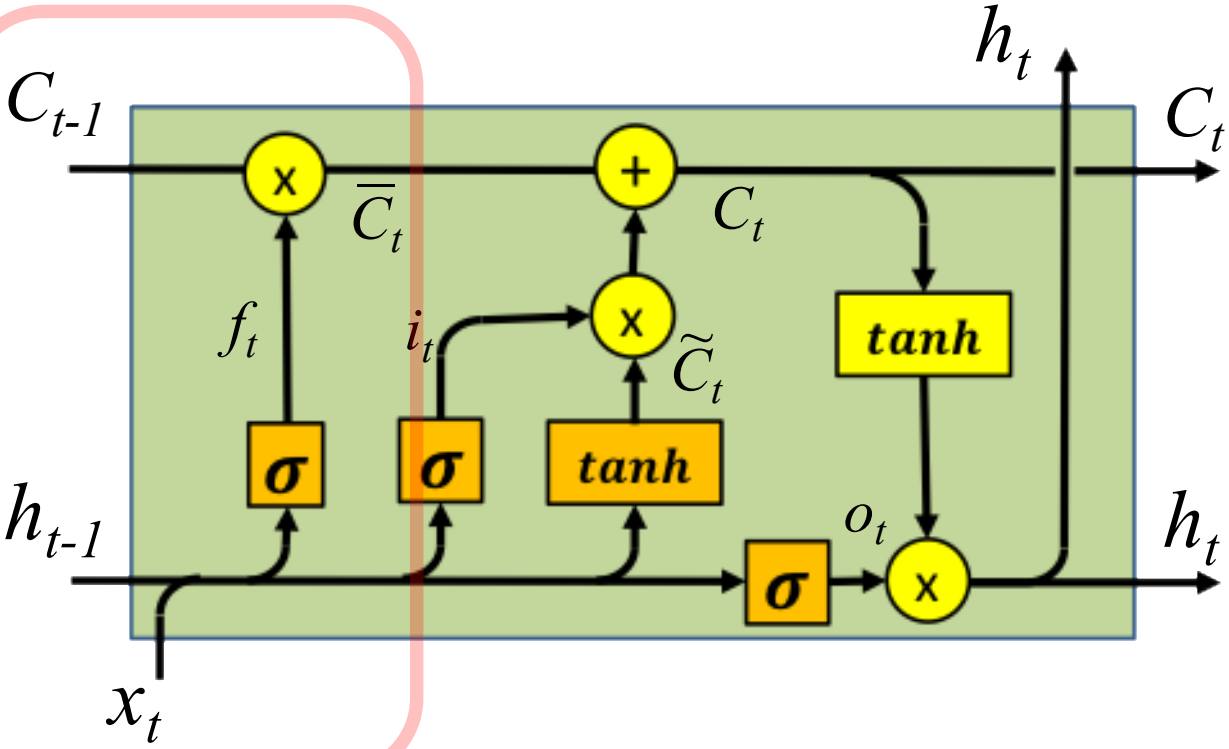
C_t : Long term memory

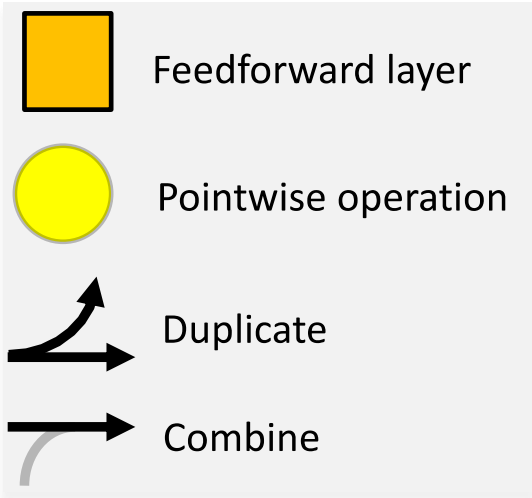
h_t : Short term memory / Output

Cell state

LSTM – How it works: forget stage

Irrelevant information is removed from the memory cell.



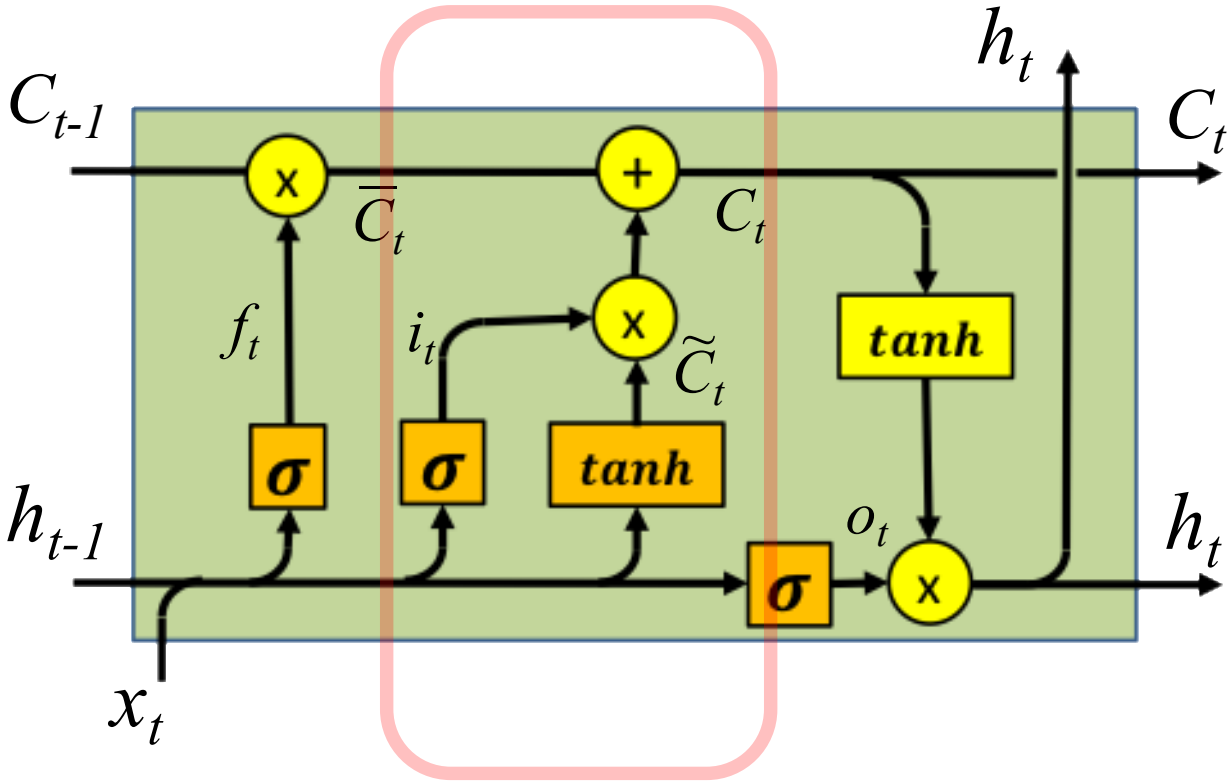


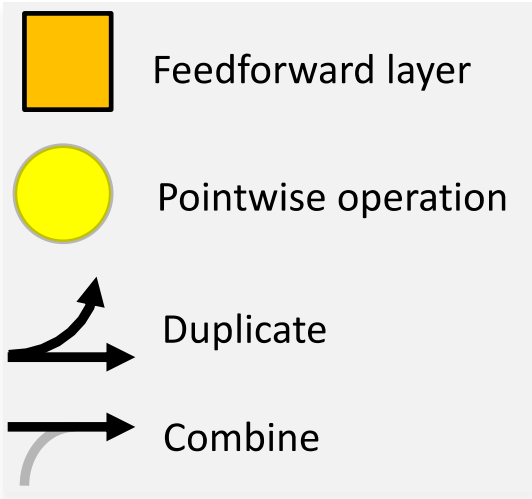
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$ - Forget gate

$$\overline{C}_t = f_t * C_{t-1}$$ - Memory cell deletion

LSTM – How it works: remember stage.

The memory cell is updated with new information.





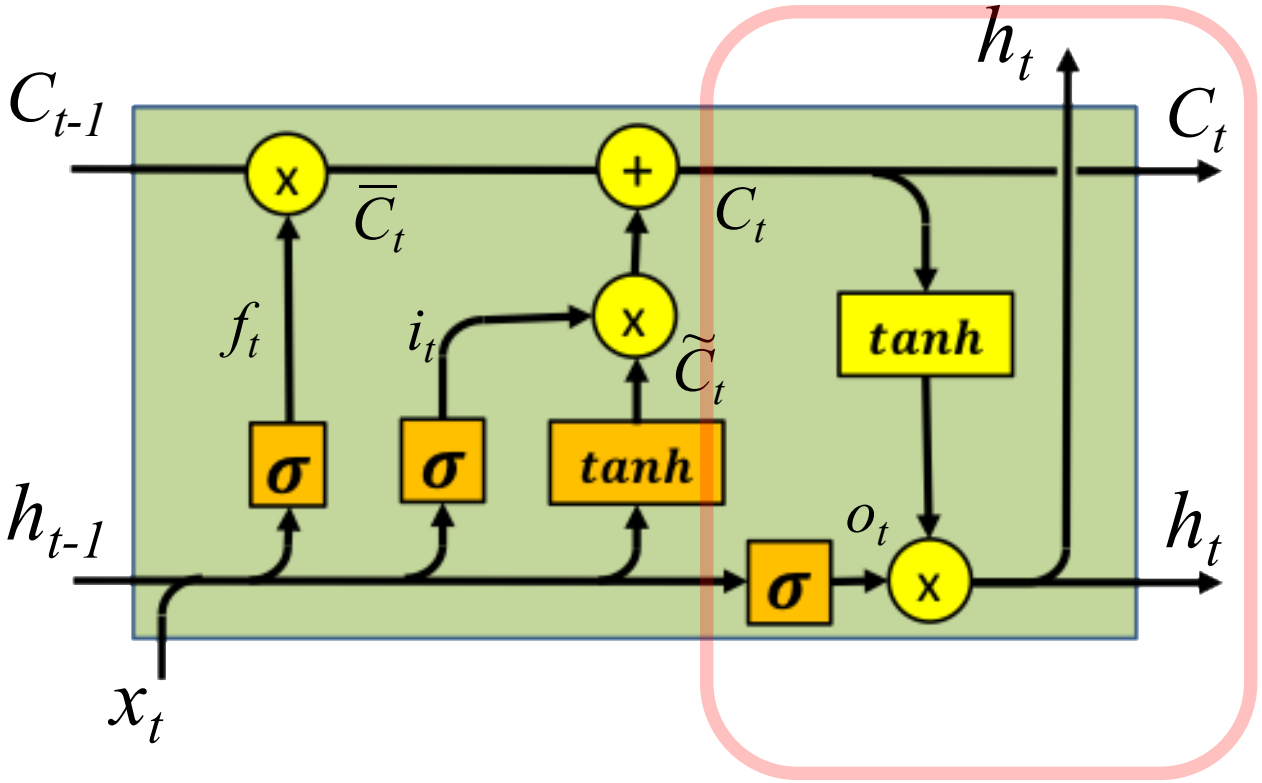
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right) \quad \text{- Input gate}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C) \quad \text{- Candidate memory}$$

$$C_t = \overline{C}_t + i_t * \tilde{C}_t \quad \text{- Memory cell update}$$

LSTM – How it works: output generation.

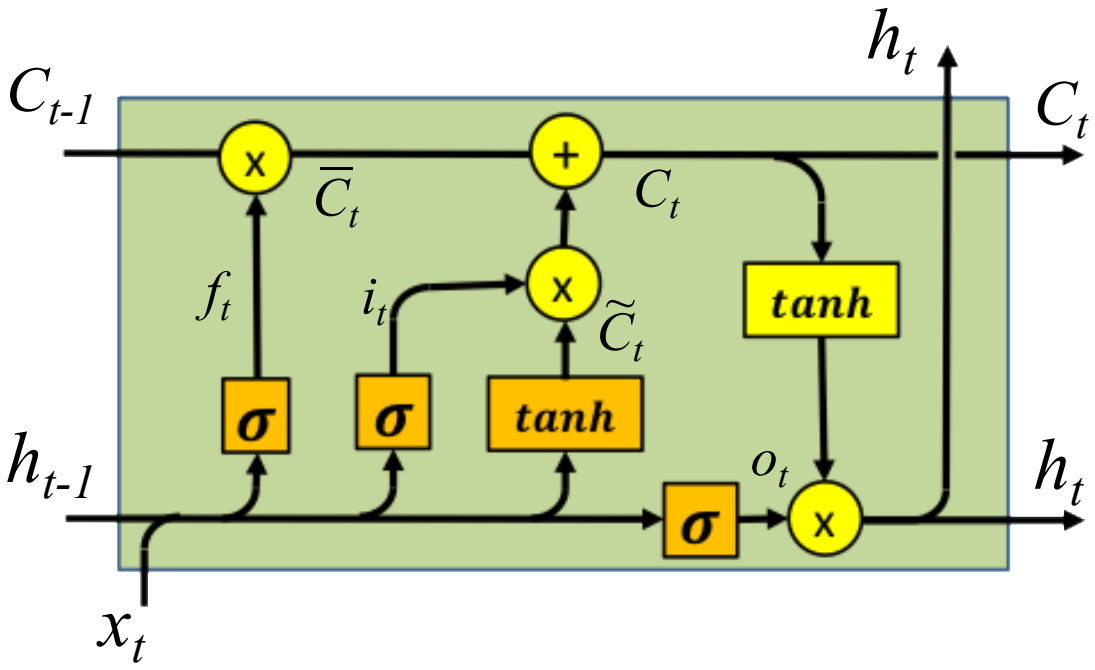
Part of the memory cell is used to generate the current output.

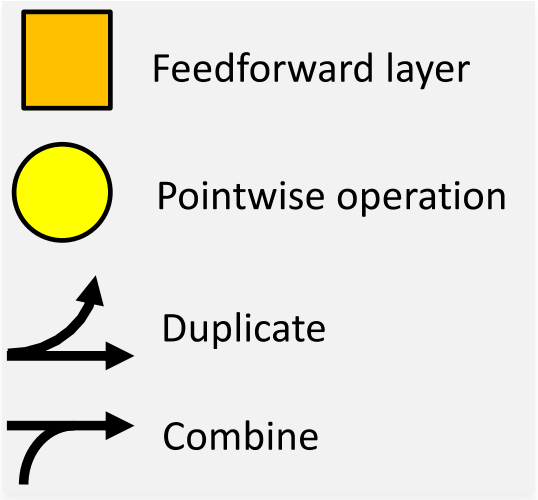


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$ - Output gate

$$h_t = o_t * \tanh\left(C_t\right)$$ - Output/Short-Term memory

LSTM – How it works: summary





$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right) \quad \text{- Forget gate}$$

$$\overline{C}_t = f_t * C_{t-1} \quad \text{- Memory cell deletion}$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right) \quad \text{- Input gate}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C) \quad \text{- Candidate memory}$$

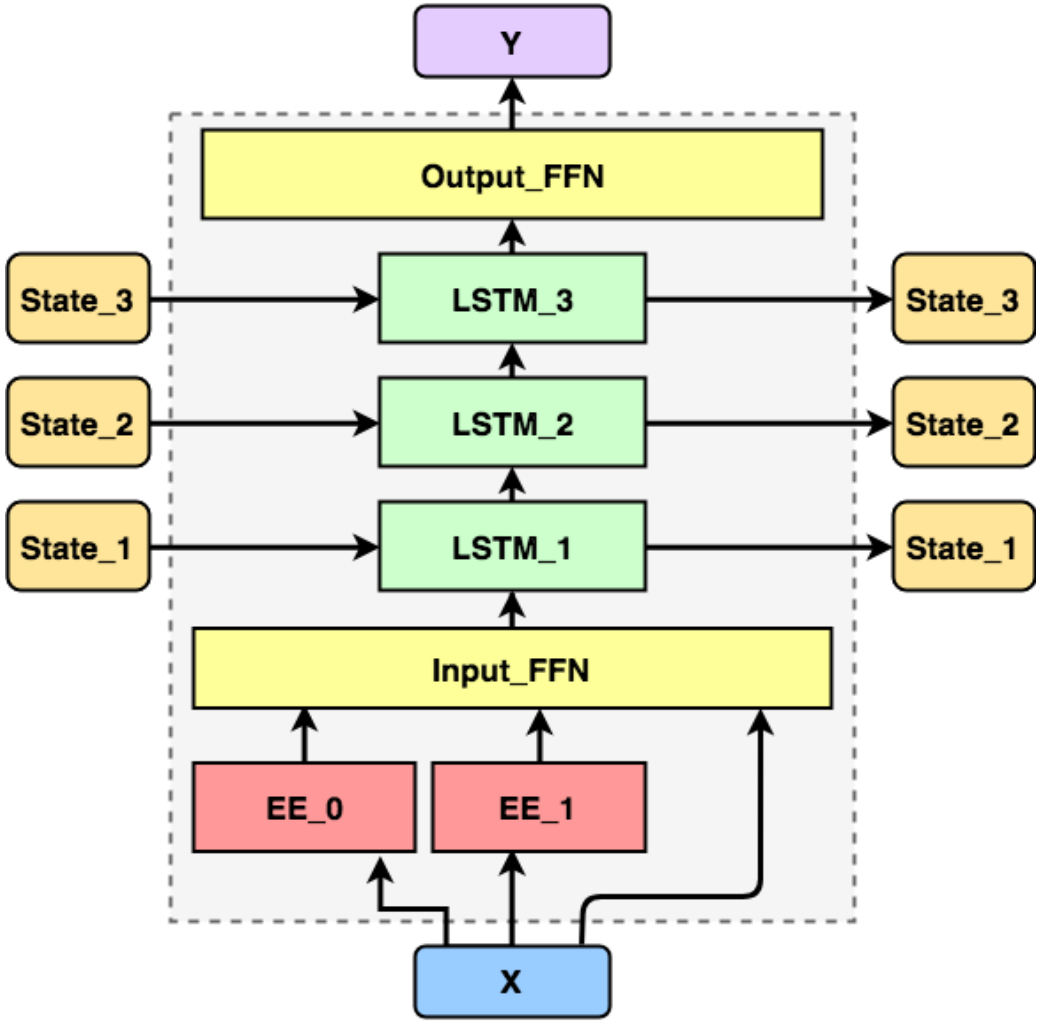
$$C_t = \overline{C}_t + i_t * \tilde{C}_t \quad \text{- Memory cell update}$$

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right) \quad \text{- Output gate}$$

$$h_t = o_t * \tanh\left(C_t\right) \quad \text{- Output/Short-Term memory}$$

RNN architecture for fraud

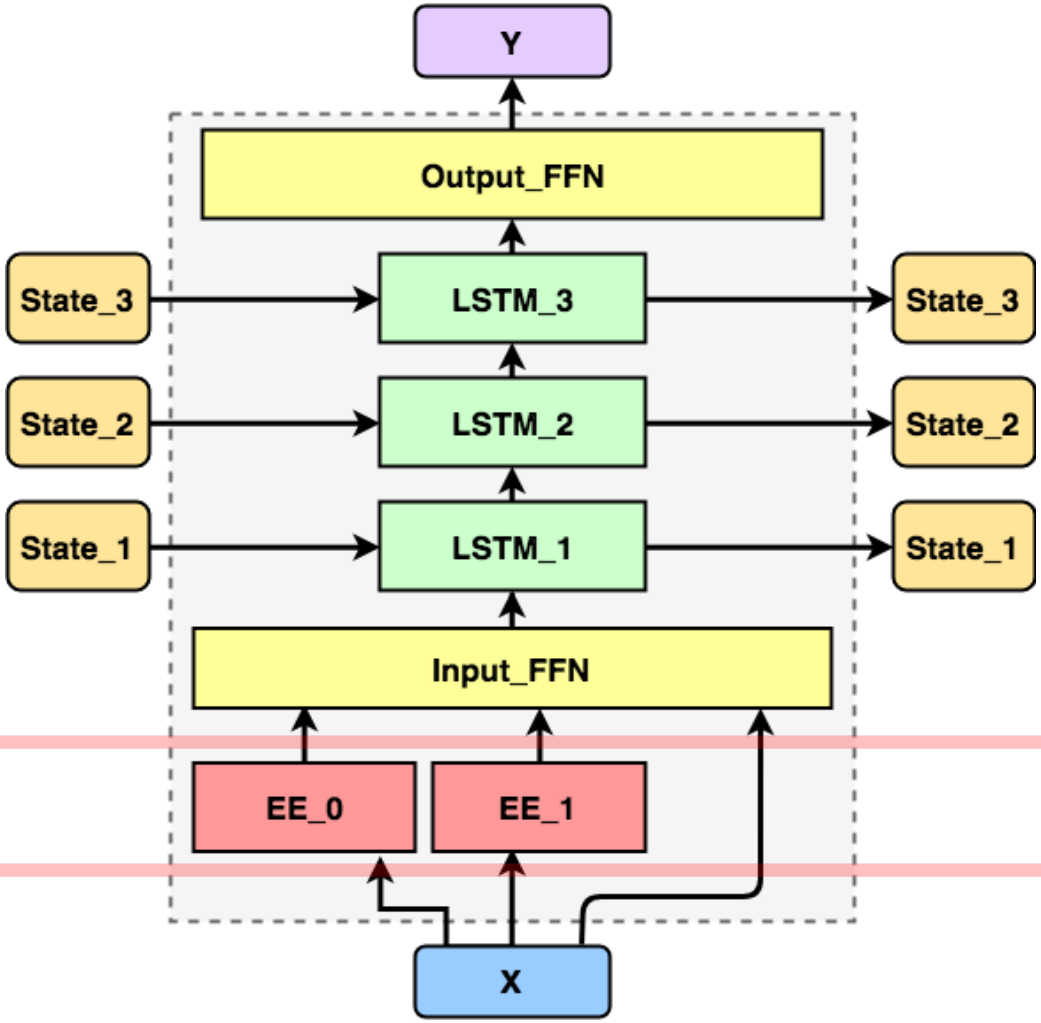
Overview.



EE : Entity Embedding Layer
FFN: Feedforward Network

Entity-embeddings(EE)

Efficiently handle categorical variables.



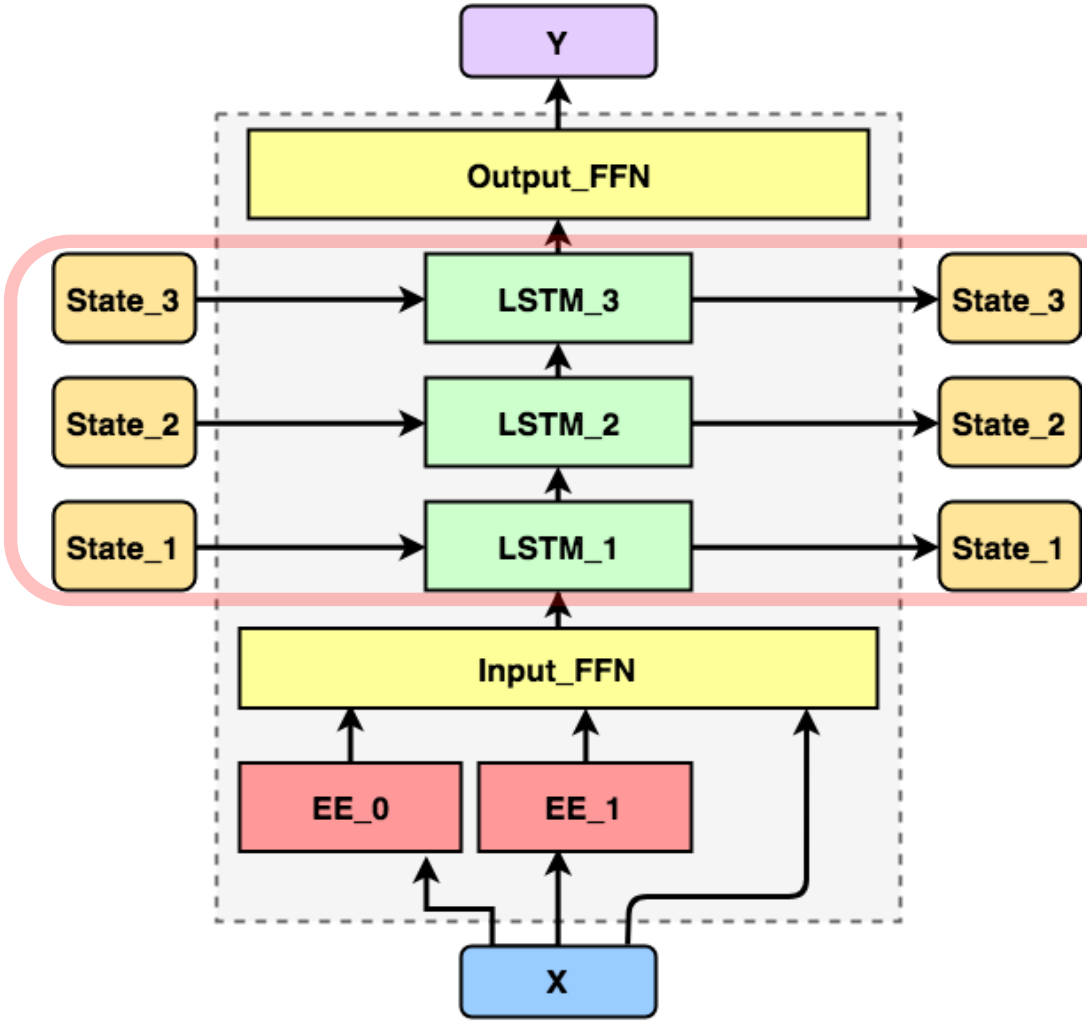
EEs are simple linear layers that map categorical variables to a lower dimensional space.

They result in a simple table look-up operation at prediction time.

Entity Embedding Layers

Multiple LSTM layers

Allow the learning complicated long term dependencies.

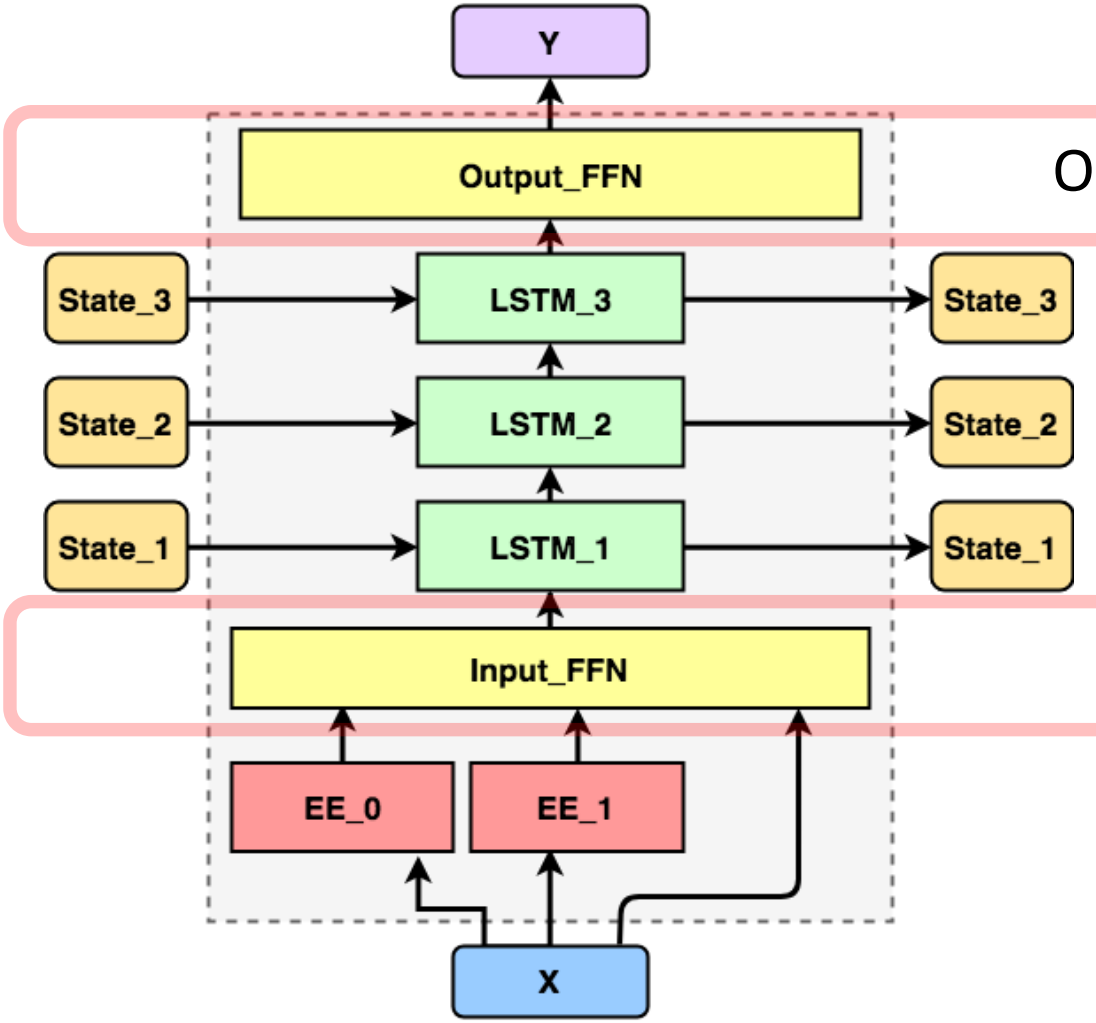


Higher level LSTM layers facilitate the learning of longer range dependencies.

LSTM layers

Feedforward input/output layers

Convert information in/out of a latent representation convenient for LSTMs.



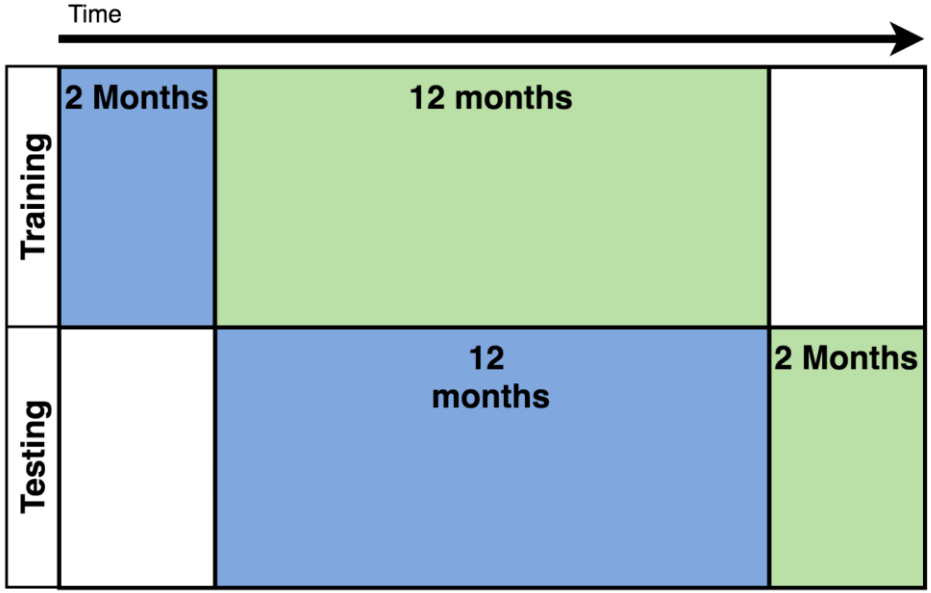
Output Feedforward Network

LSTM cells employ only single feed-forward layers internally.

Input Feedforward Network

Data Selection

Labelled data partitioning chosen to mirror GBT models as closely as possible.



Labelled transactions used for RNN/GBT training&testing.

Un-labelled transaction used by RNN.

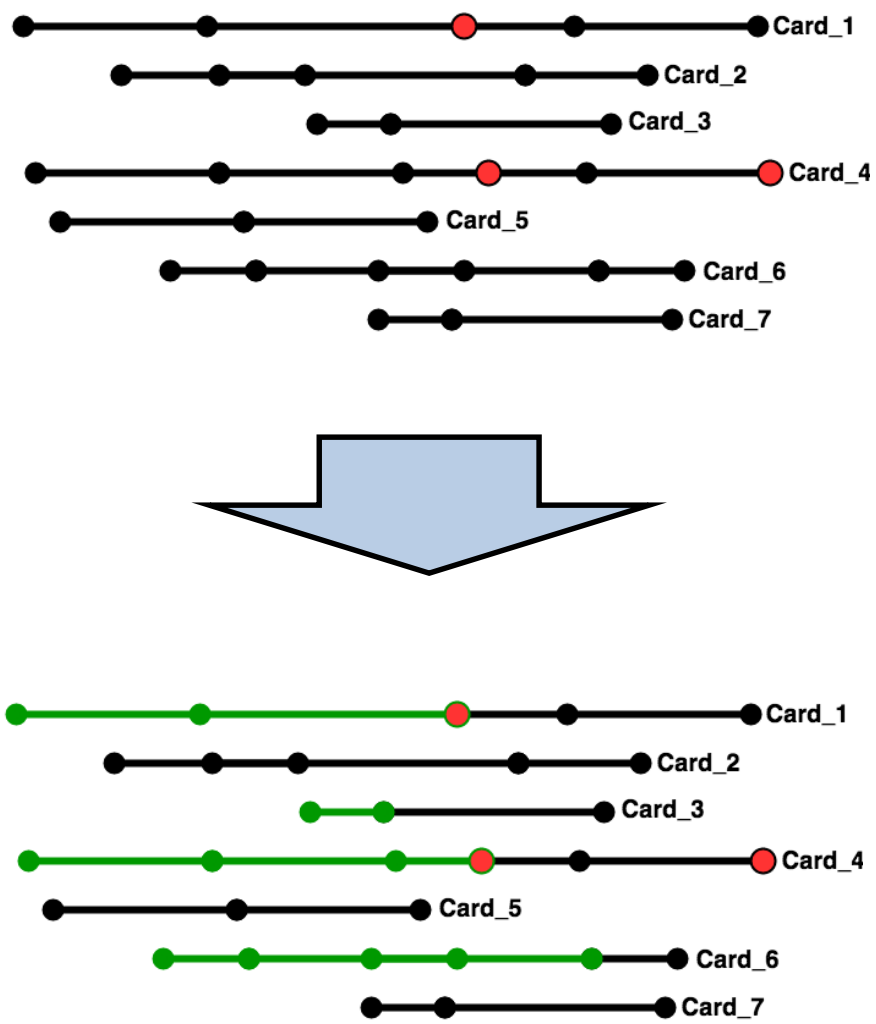
Down-sampling: we use only ~10% of the sequences that don't contain any frauds.

Features include:

- Current Transaction details.
- Some aggregations of historical transactions related to merchants still used (e.g. count, mean etc.).
- Customer details.
- We also create a 'time elapsed from previous transaction' feature to account for irregular time intervals between transactions.

Dealing with imbalanced data

Sampling and training on sub-sequences matching the desired end-label allows for precise data balancing.

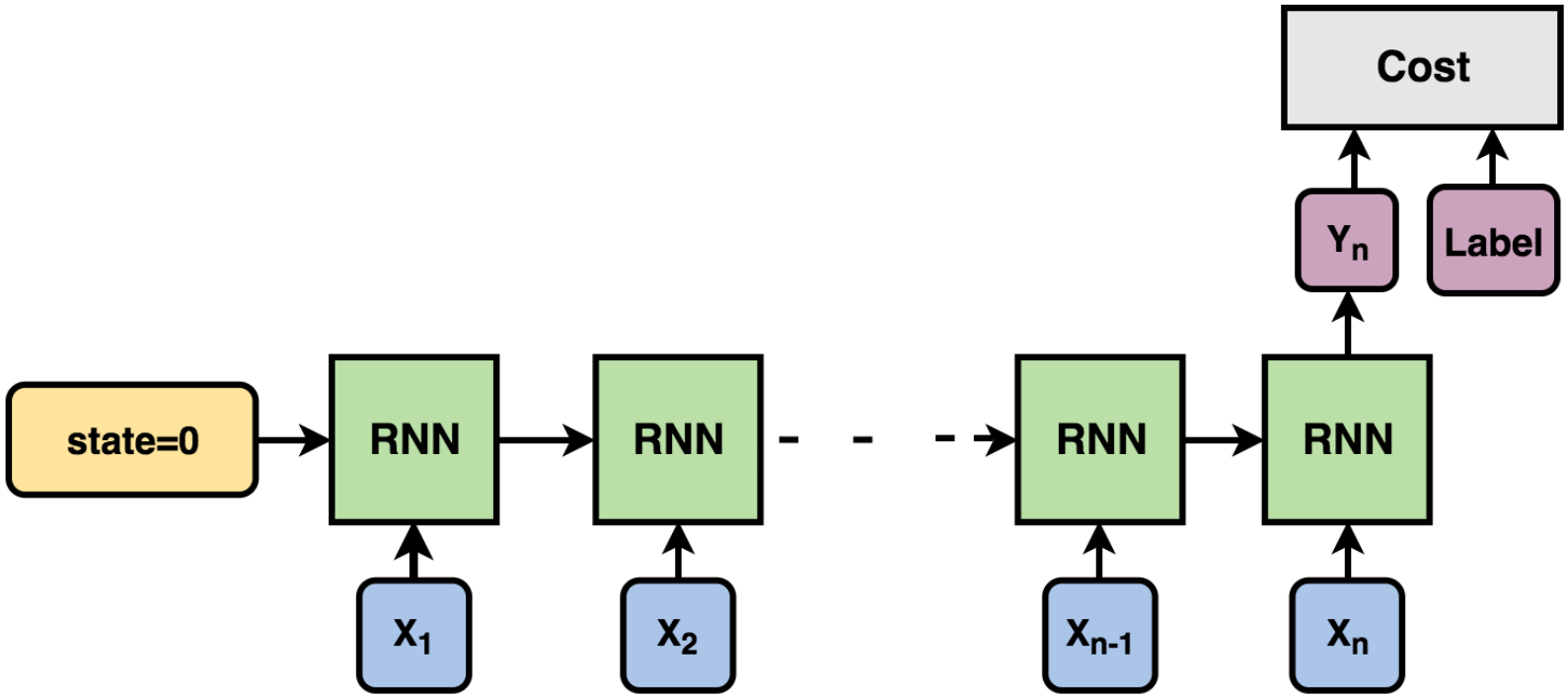


We create training batches by sampling sub-sequences ending with fraud/non-fraud with a 50/50 split.

The sub-sequences starting point is fixed.

Cost Computation

Only the sampled sequences end-label is used for training.



- Other labels will make it in the cost when part of a different sampled sub-sequence.
- Currently using cross-entropy. We have been experimenting with alternative cost functions(e.g. focal-loss, pAUC etc..)
- Scoring at run-time requires only one RNN evaluation .

Model Training

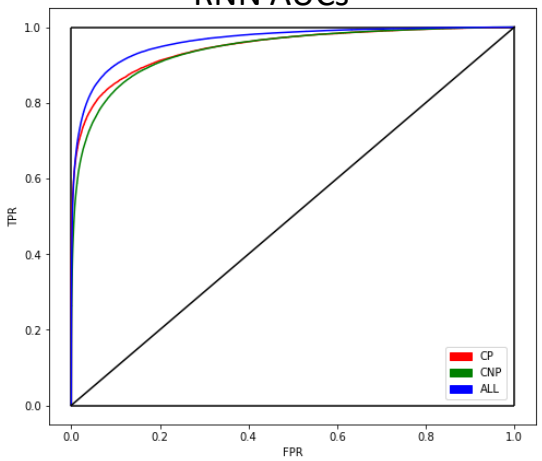
5M+ sampled sequences, ~5/6 days on a single NVIDIA P100 GPU.

- Training consists in finding the RNN parameters that minimize the chosen cost function: cross-entropy in our case.
- Model built and trained using TensorFlow. Data is fed by a proprietary streaming system.
- We employ the Adam optimizer with a learning rate exponential decay. SGD is known to find solutions that generalize better, but it is more difficult to tune…
- Hyper-parameters tuning: initial learning rate and decay rate only.

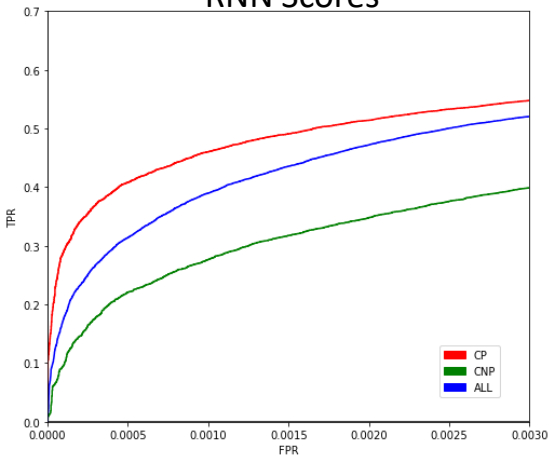
Results

Promising overall. RNN much better than incumbent model, further improvements needed to pass GBT models.

RNN AUCs



RNN Scores



$$TDR = \frac{NoOfDetectedFraudsWithinOperatingPoint}{totalFraudsInData} = TPR*100$$

TDR(Transaction Detection Rate) for ALL, CNP and CP is evaluated at different operating points.

	RNN	GBT	Incumbent
AUC ALL	0.963	NA	0.88
AUC CNP	0.94	0.94	0.76
AUC CP	0.945	0.95	0.9
TDR ALL @x bps	35%	~40%	10%
TDR CNP @y bps	34.8%	39.6%	7.5%
TDR CP @z bps	34.1%	41%	23.5%

Possible Improvements

The following improvements will likely be sufficient to surpass GBM models.

- Adopt a more consistent data selection strategy.
- Include more features (e.g. terminal, non-monetary etc.)
- Implement a hard example mining strategy.
- Use SGD rather than Adam.
- Hyper-parameters tuning.

- Most importantly: more GPUs required. This is the biggest limiting factor at the moment.

Lessons Learned

- Don’t use high level deep learning APIs, you will need total control and understanding over your model architecture and behavior. TensorFlow is well worth the effort to learn it.

- LSTMs have been employed successfully for many years over a wide range of problems. Avoid ”novel” recurrent cells, they are untested and will likely give worse results.

- Train your model until the very end, a small difference in the cost can make a big difference in the final detection rate.

- Recurrent models are notoriously slow to train and fraud is very difficult to predict. Top of the range hardware is essential, the sooner you have access to multi-GPU systems the quicker you will make progress.