Machine Learning in Blockchain Systems: AI based Consensus

Submission by:

Harsh Grover

Affiliation:

Birla Institute of Technology and Science, Pilani

Problem Statement

A need for good consensus system for blockchain is evident. That's why every new blockchain nowadays experiments with their own take on consensus protocol.

Proof of work (**PoW**) consensus system, which is used in blockchains like Bitcoin and Ethereum has numerous problems when scaled. Few of the significant ones are:

- Immense use of computing power, hence energy.
- A disadvantage to economically weak, hence it leads to centralisation in some sense.

Proof of stake (**PoS**) is another popular consensus protocol. It solves the high energy use problem of PoW. Though it enlarges the issue of centralisation and economic barrier.

A new consensus framework will be required to run blockchains in future which combines the advantages of these but minimizes their disadvantages.

Proposed Solution

Problem with both the consensus protocols discussed above arises while scaling. If we can select a subset of nodes based on certain criteria, only which will compete to create a block, this solves the issue. The created block will then be validated by all the nodes in the network. In this way, we keep the advantages of the above-stated consensus methods while eliminating their disadvantages.

I propose the selection of nodes probabilistically based on a score assigned using temporal features of node behaviour. Nodes with relatively common behaviour will have higher scores than nodes with abnormal behaviour. This is called **Proof of Commonality (PoC).**

Proof of Commonality (PoC)

First step of PoC is to assign a score to each node in the network using deep learning. The score will be given by the discriminator of a **Long Short Term Memory - Generative Adversarial Network (LSTM-GAN)** trained in an unsupervised way to learn deep temporal features of node behaviour. There will be a validated copy of this GAN at every node. The more data and parameters we use, the better discriminator will learn the normal node behaviour. We can use a function of discriminator output as the score, such that a higher score represents more common behaviour in the network.

Once assigned a score, N random numbers (between 0 and 1) are generated. Here N is the number of nodes to be selected for the competing subset. For every random number, we create a subset of nodes having a score higher than the random number. From this subset, one node is selected randomly, using another random number. Once selected, the node will get a key which can be used for the competition. This way, nodes with higher scores will have a higher probability of competing. Hence, nodes with anomalous or fishy behaviour will have less chance to participate. Afterwards, we can either use PoW or PoS with N nodes having the key. The final selected node will create the block. If validated by others, it will also perform the selection procedure for next iteration using it's validated copy of the LSTM-GAN. This way, for fake selection one should own more than half the network.

A relative common behaviour is difficult to maintain in the short term, as even by getting randomly selected for the competing list, the node becomes somewhat special/different and therefore will get a lower score in the next few selections. Hence PoC will ensure that the same nodes are not always at the top of the scoring list. This keeps the network truly decentralised.

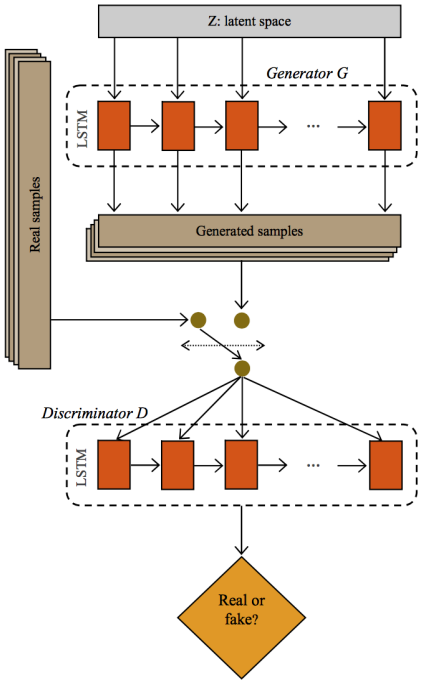
Reason for having a two-fold strategy instead of just selecting on the basis of the score is to minimize randomness and making sure that enough nodes with normal behaviour contend for building the block. This will reduce the chances of malicious nodes getting selected. The randomness also would counter any node which is trained to fool the scoring AI (discriminator) and get high scores each time. Hence in principle, the two-fold strategy makes the network more secure and decreases the number of validation failures.

Using PoC with PoW won't be that compute-intensive, as compared to just PoW, as the *difficulty* for solving the nonce won't be high in this case, due to lesser number of competing miners. It can also be combined with other low energy consuming consensus algorithms like PoS. This makes it ideal for a blockchain of IoT devices.

PoC is inspired by Proof of Involvement and Integrity (PII) and Proof of AI (PoAI). But it has a more practical implementation strategy and better decentralisation. It can potentially be implemented in current blockchains with a soft fork software patch.

Scoring

An LSTM-GAN, as shown, is trained in an unsupervised way to learn patterns from the transaction history of a node. There will be a copy of this GAN at every node, training happens via federated learning[1]. As the discriminator learns to distinguish between fake data and real data, the generator tries to produce more realistic fake data and this in turn makes discriminator better in learning qualities inherent to the real data. As not all real data have similar characteristics, the discriminator learns the most common ones. Thus we can use the discriminator of this GAN to give scores to every node in the network, corresponding to their commonality.



Data:

Data was taken from the Bitcoin network dataset**.** I downloaded the txin.dat and txout.dat files and then used a GitHub repository[2] to get a tsv file with columns: txID, in_addr, out_addr, amount (in Satoshis, i.e. 1e-8 BTC). Size of this tsv file is ~110 GB.

I extracted transaction history for every address from the generated tsv file. As the distribution of transaction amount was highly positively skewed, I scaled the values with a log function and then normalized them for better training.

To preserve temporal information as to when the transactions happened relative to each other, I used max(txID) sized array for each address. In this array, column numbers were corresponding to txIDs. Array corresponding to an address had transaction amount for the txID which that address was a part of and zeros everywhere else. Positive for credit, negative for debit.

Due to limitations in RAM provided by google colab, I couldn't load the entire file so I picked 10000 transactions from the middle. This is enough to train a prototype GAN and provide a proof of concept.

Generator (G):

One LSTM with 32 units followed by a dense layer, with hyperbolic tangent activation, to get dimensions same as real sample. Hyperbolic tangent function was used to squash the values between -1 and 1, which was the case in real samples, due to normalisation of transaction amounts.

Discriminator (D):

One LSTM with 32 units followed by a dense sigmoid layer for classification.

---

[1] A decentralised way of learning, where encrypted gradients are pushed to the cloud by each node.

[2] https://github.com/dkondor/txedges

Training:

D was pretrained for 10 epochs with binary_crossentropy loss function and default adam optimizer.

The final training followed these steps:

1. Generate shuffled batch of data with half real samples (labelled one) and half generated samples (labelled zero).
2. Set D as trainable.
3. Train D with shuffled batch.
4. Set D as non-trainable.
5. Train GD (main_model) with noise input and output labels set to one.

Repeat these steps until the generated samples start to resemble the real samples.

Loss for D in the final training was mean(y_true*y_pred).

Loss for GD was binary_crossentropy.

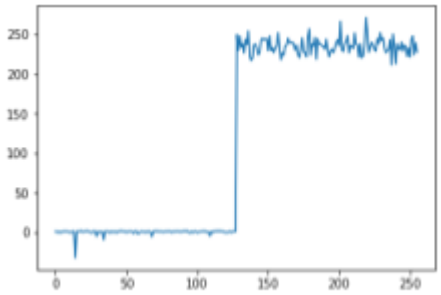
With both losses as binary_crossentropy, the model was not training properly.

Adam optimizer with learning rate 1e-5 was used.

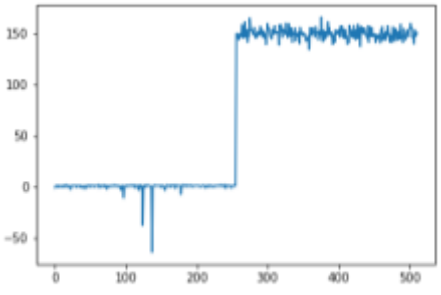
Number of epochs widely varied with starting weights of G, which were randomly initialised.

Training history:

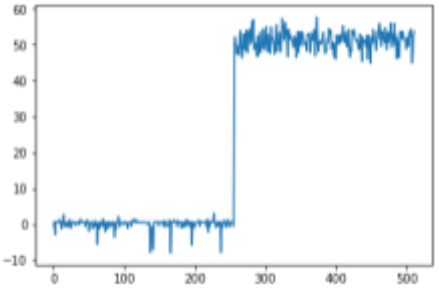
I used sum of the transaction history as a check for the resemblance of generated samples and real samples. To monitor the training I plotted this before shuffling them for training. Right half are the generated samples and left half are real samples. Y axis in each plot is the sum. Observe how the generated samples are coming closer to common real samples. This shows discriminator is aware of this, that's why generator is able to learn.



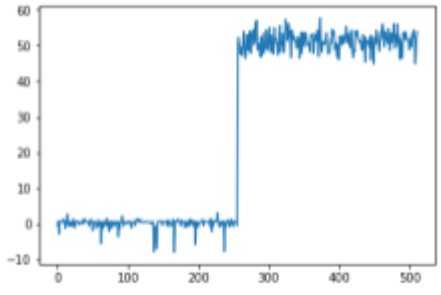
Epoch: 70



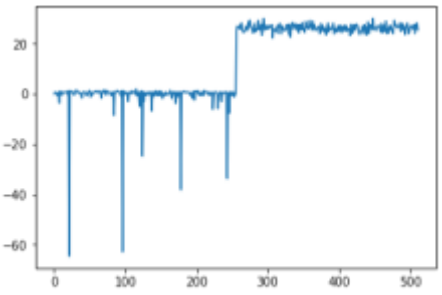
Epoch: 150



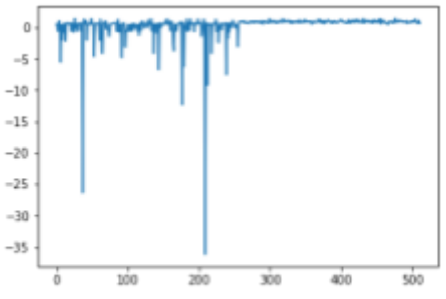
Epoch: 175



Epoch: 195



Epoch: 233

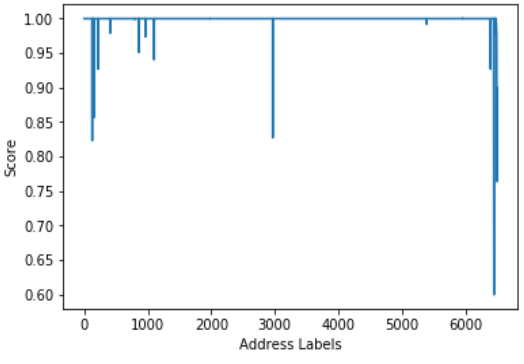


Epoch: 335

Results

The scoring function used is as follows:

```
pred = discriminator.predict(inpt_arr)
eps = 0.05
score = np.square(np.log(np.abs(pred-np.mean(pred))+eps))
score = 1-np.abs(score-np.mean(score))
```



It can be seen that most of the nodes have a high score, this should logically happen as commonality is relative. Newer nodes, i.e. ones with higher address labels, have considerably lower score which should be the case, as discussed in the next segment.

Issues

With PoC:

- PoC should ideally be used after the network is stable enough, like the current Bitcoin or Ethereum. It's fundamentally difficult to use this consensus during early stages of a blockchain.
- It can be difficult for newer nodes in the network to get a higher score as can be seen in the results. Hence they may not stay in the network for long enough to become "common". Also it becomes difficult for nodes with fishy behaviour in the past to clean their slates. To fix both of these issues we can only consider recent node behaviour, as in past 3-6 months. We can also give more weightage to more recent transactions.

With GAN:

- GANs are susceptible to adversarial attacks and can be fooled. That's why selection only on the basis of score is not fool-proof.
- Variance of the generated samples remained unchanged during the training. It was offset by 0.008 from the real samples. I tried to reduce this by training more on the converged GAN with different losses, but it quickly diverged. Better training of GAN or a more sophisticated architecture could solve this.

Scope

- The GAN I have trained is very rudimentary and just serves as a proof of concept. Only one information related to nodes were used, that is the transaction history. More information, data and features should be explored along with a more sophisticated GAN model. Only after this, the discriminator will be able to perform well in real world scenarios.
- A good scoring function should be devised which can magnify the differences in D predictions. In future work, the possibility of having different scoring mechanisms suited to different kinds of tasks to be performed by the blockchain can be looked into. This kind of consensus will make the network more efficient, especially in the case of an IoT network with different kinds of machines serving as nodes.
- PoC is a two-fold strategy. Which consensus to be used after the selection of competing nodes has to be determined and can vary based on the use-cases of the blockchain. PoC behaviour with different consensus systems should be studied.
- Planning all the intricacies of training the GAN in a decentralised way. It is also possible that the selected nodes compete by training the neural network itself. In that case, proper validation schemes have to be devised. Also that can make this consensus computationally expensive.

References

- Traditional vs Deep Learning Algorithms used in BlockChain in Retail Industry — III
- Unsupervised Approaches to Detecting Anomalous Behavior in the Bitcoin Transaction Network
- An AI Based Super Nodes Selection Algorithm in BlockChain Networks
- Consensus: Proof of Involvement and Integrity (PII)
- Coursera: Blockchain Basics
- A generic framework for privacy preserving deep learning
- Bitcoin network dataset
- https://github.com/dkondor/txedges
- Generative Adversarial Network(GAN) using Keras
- Wasserstein GAN in Keras