NPOW: A Neural Proof-of-Work

James William Fletcher
james@voxdsp.com

Abstract. A drop-in replacement for traditional proof-of-work, one that uses neural networks and machine learning. NPOW achieves this at lower power consumption, reduced reliance on mining pools, and offers miners more diversity in how they choose to solve proofs-of-work.

1. Introduction

The traditional proof-of-work released with bitcoin in 2009 has proven to be a mostly uncontested solution when it comes to ensuring each new block in a decentralised blockchain is produced by a random actor in the network [1]. While other solutions have been proposed they have either been analogues that only vary in the hash functions used or schemes that ultimately denominate to sacrificing some degree of decentralisation, such as proof-of-stake. While it is commonly contested that bitcoin mining pools can also be seen as a comparable degree of centralisation within its own right this document looks at how NPOW can alleviate if not completely disincentivise the formation of mining pools.

Furthermore, this change to proof-of-work would require little effort to implement into the bitcoin core daemon source code essentially making it a drop-in replacement, this is because NPOW retains the fundamental true random selection that traditional POW exhibits.

2. Neural Proof-of-Work

In a neural proof-of-work, miners are required to solve a set of weights to a neural network where some set of inputs have to correspond to some set of outputs. In the case of bitcoin, the inputs are prior block hashes and the outputs are the corresponding hash of the next block higher in the chain. This creates variable difficulty as the more corresponding inputs and outputs a neural network needs to learn, the more time it takes to train the network. This also ensures that each new proof-of-work is unique, as when a new block is created its new hash is added to the chain of difficulty. This is because difficulty starts at the top of the chain and works backward, the provided block weights in the proof must produce a neural network that can provide the next block hash for all blocks prior to the highest block to x iterations of difficulty. Due to the interleaved nature of training neural networks, adding just one new set of corresponding hash pairs would require a re-train of the entire neural network’s weights.

3. Implementation

The proposed topology for the neural network is simple and provides enough complexity to encode many hundreds of thousands of corresponding hash pairs to be learned. The bitcoin blockchain is currently almost 700,000 blocks high and to produce a block every 10 minutes for a high-end consumer computer today would require a difficulty of approximately 16,000 corresponding block hash pairs.

Block hashes are input into the network as hexadecimal character strings of 64 characters in length, this allows each of the 16 characters to be embedded with plenty of space between them which is used for error-correcting the output.

In C code the embedding functions…

```
#include <ctype.h>
float toEmbed(const char ic)
{
    signed char c = toupper(ic);
         if(c == '0'){return -0.8828125;}
    else if(c == '1'){return -0.765625;}
    else if(c == '2'){return -0.6484375;}
    else if(c == '3'){return -0.53125;}
    else if(c == '4'){return -0.4140625;}
    else if(c == '5'){return -0.296875;}
    else if(c == '6'){return -0.1796875;}
    else if(c == '7'){return -0.0625;}
    else if(c == '8'){return  0.0546875;}
    else if(c == '9'){return  0.171875;}
    else if(c == 'A'){return  0.2890625;}
    else if(c == 'B'){return  0.40625;}
    else if(c == 'C'){return  0.5234375;}
    else if(c == 'D'){return  0.640625;}
    else if(c == 'E'){return  0.7578125;}
    else if(c == 'F'){return  0.875;}
}

char fromEmbed(const float f)
{
         if(f >  -1          && f < -0.82421875){return '0';}
    else if(f >= -0.82421875 && f < -0.70703125){return '1';}
    else if(f >= -0.70703125 && f < -0.58984375){return '2';}
    else if(f >= -0.58984375 && f < -0.47265625){return '3';}
    else if(f >= -0.47265625 && f < -0.35546875){return '4';}
    else if(f >= -0.35546875 && f < -0.23828125){return '5';}
    else if(f >= -0.23828125 && f < -0.12109375){return '6';}
    else if(f >= -0.12109375 && f < -0.00390625){return '7';}
    else if(f >= -0.00390625 && f <  0.11328125){return '8';}
    else if(f >=  0.11328125 && f <  0.23046875){return '9';}
    else if(f >=  0.23046875 && f <  0.34765625){return 'A';}
    else if(f >=  0.34765625 && f <  0.46484375){return 'B';}
    else if(f >=  0.46484375 && f <  0.58203125){return 'C';}
    else if(f >=  0.58203125 && f <  0.69921875){return 'D';}
    else if(f >=  0.69921875 && f <  0.81640625){return 'E';}
    else if(f >=  0.81640625 && f <  1)         {return 'F';}
}
```

The neural network itself is a fully connected feed-forward network with two layers, the first has 64 inputs to 64 units and the second has 64 outputs from 64 units. The activation function used is Tanh as it exhibits the best training accuracy for this purpose. The network has a total of 8320 parameters, 128 biases, and 8192 weights *(4086 weights per layer)*. At the time of training this is 33.28 kb of float32 weights and when transported over the network in the block header is quantised to int8 at a mere 8.32 kb this is opposed to the 4-byte nonce that traditional POW currently employs. A bitcoin block is on average close to 1 megabyte so adding an 8.32 kb inflation to the block header is negligible.

For partial verification the client only needs to verify that submitted weights are not garbage, this can be achieved by doing a single forward pass checking that the highest block hash pair is correct or a simple check for a high frequency of repetitive or zero weights.

For complete verification of the proof, an iteration from the highest to the lowest corresponding block hash pairs can terminate on the first occurrence of a failed corresponding output to save compute time.

The same 256-bit integer difficulty can be reused in NPOW by dividing the current block height by the int256 maximal and then multiplying the result by

the current difficulty value whereby scaling the difficulty to the available block height range.

In a system that uses a neural proof-of-work, it is expected that many miners will produce a proof around the same time and that some multiple of them may well be valid, also because there is some added overhead to verifying neural proofs nodes may become backlogged and find it hard to pick out who provided the first valid proof. Employing a policy of allowing one proof submission between blocks per-peer should help alleviate this but ultimately the entire network will default to the selection by the majority of the network. Block intervals are scaled to 10-minute intervals by the difficulty which should give ample time for most of the network to process the first winning block.

4. Mining Pools

Mining Pools can still exist but they are heavily disincentivised. This is because in an NPOW network a majority of miners will come to a solution towards the end of a 10 minute block period whereas in traditional POW it is expected a miner will find a solution no sooner than the end of the 10 minute block period. In POW a solution may be found typically between 8 and 50 minutes by a single miner and in NPOW many miners will come to a solution at varying times soon after or just before the 10 minute block period. This means that in NPOW the delay in propagating the winning block by sending it via a pool first could significantly impact the chances of winning.

Furthermore, it would be difficult for a pool to decide how to fairly split up the winning reward. A pool would need to take into account how soon each proof had been submitted and what percentage of corresponding hash pairs each proof got correct for each miner.

5. Solo Mining

Miners would typically be running a python mining program using a parallelised back end such as Tensorflow. Miners will be unlikely to verify their submissions before submitting them due to the tight competition on submitting their proof first, as such miners would submit their weights in blind faith that the number of training iterations they used was sufficient. Submitting weights using int8 quantisation further helps to increase the entropy of the correctness of the submitted proof and thus who submits the first winning block.

Mining can be parallelised over multiple GPUs but would gain little incentive over parallelised CPUs. The cost to benefit ratio is currently tight as a single 400 GBP GPU comes in at around ~10% more performant than an equivalent 200 GBP CPU when training with Tensorflow as of 2021. Considering that a GPU would also need to be connected to a computer with a CPU the benefit would come down to hardware cost, power consumption, and how many GPUs can a user chain to a single motherboard.

6. Streak Mining

One side effect of this system is that the producer of a valid proof could have a head start in producing the next block as that producer having produced the next block hash can potentially know the next block hash before anyone else. I refer to this vector as streak mining and the main disruptor to streak mining is that the verification of the trained network is a lengthy process, miners don’t know that they have the next block hash until they verify their model and a model that takes 10 minutes to train can take almost as long to verify, by which time other miners could be well into training their weights for the new block because miners would have to submit the weights before verification to have any chance in submitting the first winning proof.

7. Power Consumption

In traditional POW a miner works tirelessly to find solutions to blocks, and while this may seem very much the same for NPOW there can be periods of rest between submitting proofs unless the miner spends that time post-verifying the submission to decide if they have time to train and submit another attempt, which is going to be rare if at all occurrent. Thus periods of rest are to be frequently expected between block submissions, also the workload to train a neural network is arguably less intensive than a brute force hash loop in most cases because traditional proof-of-work miners can take better advantage of memory caching than a neural network can which means that the CPU load when training a neural network will generally be lower as it becomes bottlenecked by memory bandwidth depending on the batch size used in the back-propagation.

Ultimately the extra flexibility miners are given on how they compute their network weights albeit fixed to using the Tanh activation function means that there is further room for improvement over the power consumption of neural proofs-of-work in the near future. Whereas the future power consumption of traditional POW is only improved by increasing the power efficiency of the hardware running the mining algorithm.

8. Conclusion

This is a neural proof-of-work scheme that is analogous to the traditional proof-of-work bitcoin currently uses which makes the implementation simplified as NPOW can make use of existing features such as difficulty without any explicit changes, and still provides the same true random selection over who submits the next block in the chain. Miners are given more flexibility which comes with positive trade-offs to power consumption and the increased block header size and verification time is negligible to the 10-minute block interval or network bandwidth/block size.

References

[1] Satoshi Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System ”, https://bitcoin.org/bitcoin.pdf, 2008