Summary of Formulas

Validator–Miner Mechanism in the Moderntensor Project

Moderntensor Development Team

January 8, 2025

Introduction

This document describes the calculation of **incentives**, **scores** for *miners/validators*, the **penalty** and **recovery** mechanisms when fraud is detected, as well as how to calculate the **weight** for validators in a decentralized *AI* system similar to **Bittensor**.

Additionally, the last section extends the document by discussing:

- Direct slashing of the **validator's stake** in case of fraud.
- **Incentives** for validators (similar to miners).
- A more **logical multi-layered consensus/approval mechanism**.

Key Variables and Constants (e.g.):

- m, n: Number of *miners*, number of *validators* (depending on convention).
- i, j: Index for *miners*, *validators* (or vice versa).
- $\theta, \alpha, \beta, \gamma, \lambda$: Coefficients, thresholds, recovery speeds, etc.
- Q_{task}: Task completion ratio.
- D_{miner}: Approval ratio for *miners*.
- $E_{\text{validator}}$: Performance score (or trust score) for *validators*.
- $W_{\text{validator}}$: Weight of the *validator*, combining **stake** and **performance**.

Recommendation: It is advised to add a *glossary* or *section* describing the meaning and scope of the variables (i, j), to avoid confusion during code implementation.

1. Incentive Formula for Miner

Suppose you want to distribute the reward (*incentive*) for **miners** based on their contribution relative to the total network:

$$\text{Incentive\_miner}(x) = \frac{\sum_{j=0}^{m}\Big(W_x \times P_{xj}\Big)}{\sum_{i=0}^{n}\sum_{j=0}^{m}\Big(W_i \times P_{ij}\Big)},$$

where:

- x is the index of the *miner* under consideration.
- W_x is the **weight** (or stake) of *miner* x.
- P_{xj} is the *performance* (or approval level) of *miner* x interacting with *validator* j.

The denominator is the total contribution of all *miners* in the network (or subnet).

2. Task Index Q_{task}

The task completion ratio (can be computed for the subnet or per individual *validator*):

$$Q_{\text{task}} = \frac{\sum(\text{task\_success})}{\sum(\text{task})}.$$

Here, $\sum(\text{task\_success})$ = total successful tasks, $\sum(\text{task})$ = total tasks assigned.

3. Miner Evaluation D_{miner}

D_{miner} reflects the "approved ratio" of a *miner* i. For example:

$$D_{\text{miner}}(i) = \frac{\sum(\text{approved\_i\_by\_validators})}{\text{total number of checks with miner } i}.$$

A *miner* with D_{miner} **too high** (above the D_{upper} threshold) could be suspected of "being indiscriminately approved by validators".

4. Performance Score of Validator $E_{\text{validator}}$

The performance score (or "trust score") of a *validator*:

$$E_{\text{validator}} = \theta \times Q_{\text{task}} \; + \; (1-\theta) \times D_{\text{miner}},$$

- $\theta \in [0,1]$: The coefficient determining the relative importance of Q_{task} and D_{miner}.
- Q_{task}: The task completion ratio of the *validator*.
- D_{miner}: Reflects the quality of *miners* that the *validator* approves.

5. Penalty Mechanism for "Fraudulent" Validators

When a *validator* exhibits fraudulent behavior (indiscriminate approval, arbitrary rejection), we adjust:

$$P_{\text{adjust}} = \begin{cases} 1 - \alpha\Big(D_{\text{miner}} - D_{\text{upper}}\Big), & \text{if } D_{\text{miner}} > D_{\text{upper}}, \\ 1 - \beta\Big(D_{\text{lower}} - D_{\text{miner}}\Big), & \text{if } D_{\text{miner}} < D_{\text{lower}}, \\ 1, & \text{if } D_{\text{lower}} \leq D_{\text{miner}} \leq D_{\text{upper}}. \end{cases}$$

$$E_{\text{validator\_new}} = E_{\text{validator\_base}} \times P_{\text{adjust}}.$$

If repeat offenses occur, gradually increase $\alpha \leftarrow \alpha + \Delta\alpha$, $\beta \leftarrow \beta + \Delta\beta$.

6. Recovery of Emission for "Fraudulent" Validators

When a *validator* maintains **good behavior** (keeps D_{miner} in $[D_{\text{lower}}, D_{\text{upper}}]$), their score can be **gradually restored**:

$$E_{\text{validator\_restore}} = E_{\text{validator\_new}} \; + \; \gamma\Big(E_{\text{validator\_base}} - E_{\text{validator\_new}}\Big).$$

If the result exceeds $E_{\text{validator\_base}}$, then set $E_{\text{validator\_restore}} = E_{\text{validator\_base}}$.

7. Calculating Validator Weight $W_{\text{validator}}$

To consider both **stake** and **performance**:

$$W_{\text{validator}} = \lambda \times \frac{\text{stake\_validator}}{\sum(\text{stake\_subnet})} \; + \; (1 - \lambda) \times E_{\text{validator}},$$

- $\lambda \in [0, 1]$: The coefficient prioritizing between **stake** and **performance**.
- stake_validator: The stake of the validator.
- $\sum$(stake_subnet): The total stake of all validators in the subnet.

8. Miner Performance with Each Validator

To evaluate performance (or the miner–validator pair) across tasks:

$$P_{\text{miner\_edge}}(v) = \frac{\sum_{j=0}^{n}\Big(W_{\text{task}}(j) \; \times Q_{\text{task}}(j, \; v)\Big)}{\sum_{j=0}^{n} W_{\text{task}}(j)},$$

- $W_{\text{task}}(j)$: Weight of task j.
- $Q_{\text{task}}(j, v)$: Success ratio of validator v on task j (or for a specific miner).

9. Suggestions for Improvements Optimization

1. Direct Slashing into Stake

- When a validator *commits serious violations*, **slashing the stake** could be applied.
- Increases financial risk if fraud persists.
- The slashed stake can be **burned** or put into the reward pool (*emission pool*).

2. Sharing Incentives for Validators

- Similar to "`Incentive_miner`", an "`Incentive_validator`" can be implemented.
- Reward validators based on $E_{\text{validator}}$ or correct decision ratios.
- Distribute **emission pool**: (x% for miners, y% for validators).

3. Lock-up Period for Stake

- Require a certain *lock-up* time to prevent "stake - fraud - unstake" schemes.
- For example: After staking, need to wait N blocks/epochs before unstaking.

4. Simulation Stress Testing

- Write **simulation** to model hundreds/thousands of nodes.
- Test convergence speed, penalty probability, etc.

5. Off-chain Aggregator Batch Updates

- Perform *score* calculations off-chain, then only push *hash* or *Merkle root* to the chain.
- Reduce transaction fees, **batch multiple updates** in a single transaction.

6. Logging, Replay Attack, Transparency

- Store **logs** of each update, with block time.
- Use **nonce** or **sequence** to prevent replay attacks.

7. Decentralized Governance (DAO)

- Allow **voting** to change parameters $\alpha, \beta, \gamma, \lambda, \theta, \ldots$
- Create/terminate subnets, adjust reward distribution ratios, etc.

10. Extended Document Section

In addition to the improvement proposals, you can add **the following topics** to build a more comprehensive *decentralized AI* system:

A. Direct Slashing into Stake

- Instead of just reducing the performance score, a *validator* committing **multiple frauds** will have their stake directly slashed.
- It could apply **linear slashing** (percentage of stake slashed) or **progressive slashing** (increased penalty if repeated).

Meaning: Helps provide a stronger **financial disincentive** to reduce intentional bad behavior.

B. Incentives for Validators

- Similar to the `Incentive_miner` calculation, you can design `Incentive_validator`:

$$\text{Incentive\_validator}(v) = \frac{W_v \times E_v}{\sum_{u \in V}(W_u \times E_u)},$$

- Split the common *emission* into two branches: Miner Incentive, Validator Incentive (e.g., *70%-30%*, etc.).

Meaning: Creates **motivation** for validators to act honestly, contributing to network security.

C. Multi-layered Consensus or Approval Mechanism

- Instead of just one layer of validators, you can have multiple **layers** (level of approval):
  - Layer 1: Basic validation (check signatures, formats, etc.).
  - Layer 2: AI quality (score, correctness).
  - Layer 3: On-chain consensus (voting or oracle).
- Each layer can have its own **penalty/reward** parameters, which can be **decentralized** or semi-centralized.

Meaning: Increases **trustworthiness** and **scalability** in complex AI systems.

Conclusion

The document has thoroughly presented the core **formulas** (scoring, rewards/penalties, recovery, etc.) and **improvement suggestions** for building a **decentralized AI system** that is sustainable, similar to Bittensor.
Summary:

- Regularly *simulate*, *stress test* to find the optimal values for parameters like $\theta, \alpha, \beta, \lambda, \ldots$
- Build a flexible **governance** (DAO) system, ready to adjust stake, incentives, and approval logic.
- Ensure **security** (nonces, logs), **prevent replay attacks**, and maintain transparency on-chain.

Next Steps:

1. *Deploy* on Cardano (or another blockchain platform), program **smart contracts** (Plutus, Substrate, EVM, etc.).
2. Integrate the **AI module** (off-chain aggregator, NN model) and the **stake manager** (lock-up period, slashing).
3. Expand the **community** participation, apply the **voting** mechanism to upgrade the system.

. . . End . . .