Bittensor: A Peer-to-Peer Intelligence Market

Yuma Rao

www.bittensor.com

Abstract

As with other commodities, markets could help us efficiently produce machine intelligence. We propose a market where intelligence is priced by other intelligence systems peer-to-peer across the internet. Peers rank each other by training neural networks which learn the value of their neighbors. Scores accumulate on a digital ledger where high ranking peers are monetarily rewarded with additional weight in the network. However, this form of peer-ranking is not resistant to collusion, which could disrupt the accuracy of the mechanism. The solution is an incentive mechanism which maximally rewards honestly selected weights, making the system resistant to collusion of up to 50 percent of the network weight. The result is a collectively run intelligence market which continual produces newly trained models and pays contributors who create information theoretic value.

The production of machine intelligence has come to rely almost entirely on a system of benchmarking, where machine learning models are trained to perform well on narrowly defined supervised problems. While this system works well for pushing the performance on these specific problems, the mechanism is weak in situations where the introduction of markets would enable it to excel. For example, intelligence is increasingly becoming un-tethered from specific objectives and becoming a commodity which is (1) expensively mined from data (Schwartz et al. [2019]), (2) monetarily valuable (OpenAI [2020]), (3) transferable (Devlin et al. [2019]), and (4) generally useful (Radford et al. [2019]). Measuring its production with supervised objectives does not directly reward the commodity itself and causes the field to converge toward narrow specialists (Chollet [2019]). Moreover, these objectives (often measured in uni-dimensional metrics like accuracy) do not have the resolution to reward niche or legacy systems, thus what is not currently state of the art is lost. Ultimately, the proliferation of diverse intelligence systems is limited by the need to train large monolithic models to succeed in a winner-take-all competition. Standalone engineers cannot directly monetize their work and what results is centralization where a small set of large corporations control access to the best artificial intelligence (OpenAI [2020]).

A new commodity needs a new type of market[1]. This paper suggest a framework in which machine intelligence is measured by other intelligence systems. Models are ranked for informational production regardless of the subjective task or dataset used to train them. By changing the basis against which machine intelligence is measured, (1) the market can reward intelligence which is applicable to a much larger set of objectives, (2) legacy systems can be monetized for their unique value, and (3) smaller diverse systems can find niches within a much higher resolution reward landscape. The solution is a network of computers that share representations with each other in a continuous and asynchronous fashion, peer-to-peer (P2P) across the internet. The constructed market uses a digital ledger to record ranks and to provide incentives to the peers in a decentralized manner. The chain measures trust, making it difficult for peers to attain rewards without providing value to the majority. Researchers can directly monetize machine intelligence work and consumers can directly purchase it.

[1]"The iron rule of nature is: you get what you reward for. If you want ants to come, you put sugar on the floor." - Charlie Munger

1 Model

We begin with an abstract definition of intelligence Hinton et al. [2015] in the form of a parameterized function $y = f(x)$ trained over a dataset $D = [X, Y]$ to minimize a loss $\mathcal{L} = E_D[\mathcal{Q}(\ y,\ f(x))\ )]$. Our network is composed of n functions $F = f_0, ..., f_j, ...f_n$, 'peers' where each is holding zero or more network weight $\mathbf{S} = [s_i]$ 'stake' represented on a digital ledger. These functions, together with losses and their proportion of stake, represent a stake-weighted machine learning objective $\sum_i^n \mathcal{L}_i * s_i$.

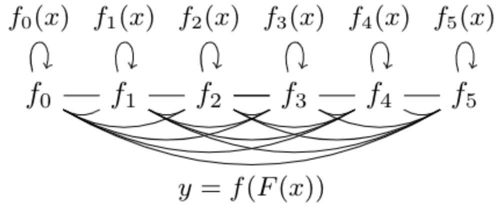
$$\begin{tikzcd}
\mathcal{L}_0 & \mathcal{L}_1 & \mathcal{L}_2 & \mathcal{L}_3 & \mathcal{L}_4 & \mathcal{L}_5 \\
f_0 \arrow[u] & f_1 \arrow[u] & f_2 \arrow[u] & f_3 \arrow[u] & f_4 \arrow[u] & f_5 \arrow[u] \\
D_0 \arrow[u] & D_1 \arrow[u] & D_2 \arrow[u] & D_3 \arrow[u] & D_4 \arrow[u] & D_5 \arrow[u]
\end{tikzcd}$$

Figure 1: Peer functions with losses $\mathcal{L}_i$ and unique datasets D_i.

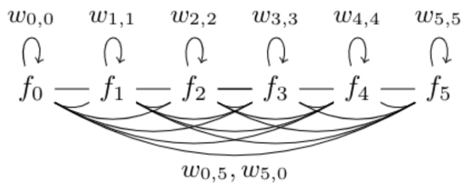
Our goal is the distribution of stake I, as incentive, to peers who have helped minimize the loss-objective (Figure-1), and importantly, in such a way that, it is difficult for a small proportion of stake to collude as a means to maximize their distribution in the network without minimizing the loss (Figure-3).

$$\mathbf{S}_{t+1} = \mathbf{S}_t + \tau \mathbf{I} \tag{1}$$

In this paper, we suggest this can be achieved through *peer-ranking*, where peers use the outputs of others $F(x) = [f_0(x)...f_n(x)]$ as inputs to themselves $f(F(x))$ and learn a set of weights $\mathbf{W} = [w_{i,j}]$ where peer i is responsible for setting the i^{th} row through transactions on a digital ledger.


$f_0(x)$ $f_1(x)$ $f_2(x)$ $f_3(x)$ $f_4(x)$ $f_5(x)$
f_0 f_1 f_2 f_3 f_4 f_5
$y = f(F(x))$

(a) Inter-function connectivity.


$w_{0,0}$ $w_{1,1}$ $w_{2,2}$ $w_{3,3}$ $w_{4,4}$ $w_{5,5}$
f_0 f_1 f_2 f_3 f_4 f_5
$w_{0,5}, w_{5,0}$

(b) Weight matrix.

Setting weights using an fishers information pruning score LeCun et al. [1989]; Yu et al. [2017] in the ranking calculation, $\mathbf{R} = \mathbf{W}^T \cdot \mathbf{S}$, achieves an idealized scoring where each peer's incentive is equivalent to its pruning score: *the cost in entropy towards* $\sum_i^n \mathcal{L}_i * s_i$ *induced by removing it from the network.*

$$r_i \approx \frac{1}{n} \sum_j^n \sum_{x \in D_j} \Delta F^T(x)_i \cdot \mathbf{H}(\mathcal{Q}_j(x)) \cdot \Delta F(x)_i \tag{2}$$

However, this approach is not resistant to collusion, where peers vote for themselves, notably instead of using (2), and set weights to enhance their own inflation at the expense of the network(Figure-3). This attack is trivial since the digital ledger cannot audit the parameters of each model, only the inter-model weights $\mathbf{W}$.

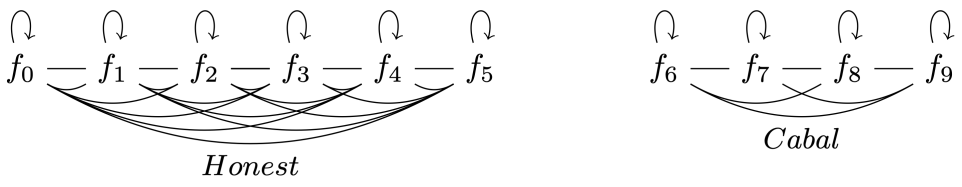


Figure 3: Disjoint cabal: peers in the right sub-network only vote for themselves.

2 Incentive

We extended the naive ranking method to evade collusion with an 'incentive' function $I(W, S)$ which limits reward to peers which have not reached consensus in the network. Assuming no group of peers holds more than the majority of stake in the system, then peers can only attain inflation by working to attract votes from the majority: a core assumption in many decentralized system like Bitcoin.

Reintroducing our terms, our incentive mechanism requires a stake vector $\mathbf{S}$ and a set of weights $\mathbf{W}$ where rows are inter-peer rankings. We also infer a trust matrix $\mathbf{T}$ from the weights, where $t_{i,j} = 1$ if and only if there is a non-zero edge between peer i and j.

$$\mathbf{W} = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} \\ w_{1,0} & w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,0} & w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,0} & w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix} \quad \mathbf{S} = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} t_{0,0} & t_{0,1} & t_{0,2} & t_{0,3} \\ t_{1,0} & t_{1,1} & t_{1,2} & t_{1,3} \\ t_{2,0} & t_{2,1} & t_{2,2} & t_{2,3} \\ t_{3,0} & t_{3,1} & t_{3,2} & t_{3,3} \end{bmatrix} \tag{3}$$

We define peers who have reached 'consensus' as those with non-zero edges from more than 50 percent of stake in the network. (This is simply the normalized values of $(T^T \cdot S) > 0.5$). To ensure the mechanism differentiable we define this computation using the continuous sigmoid function. The sigmoid produces a threshold-like scaling which rewards connected peers and punishes the non-trusted. The steepness and threshold point can be modulated through a temperature ρ and shift term κ.

$$\mathbf{C} = \sigma(\rho(\mathbf{T}^T \mathbf{S} - \kappa)) \tag{4}$$

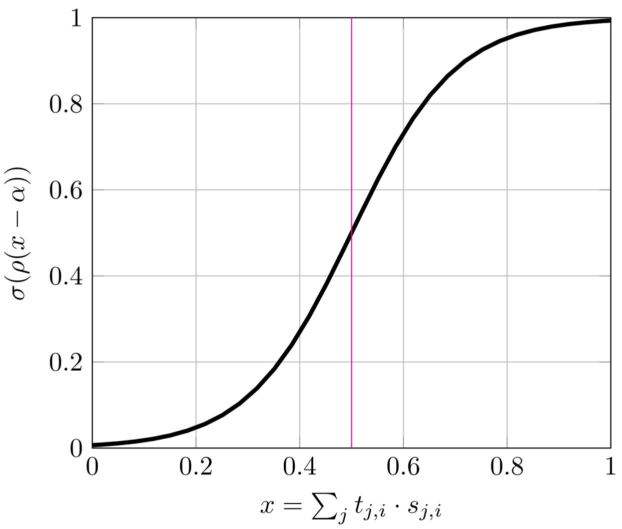


Figure 4: Consensus function $c_i = \sigma(\rho \sum_j t_{j,i} s_j - \kappa)$ with temperature $\rho = 10$ and shift $\kappa = 0.5$. The activation takes the trust scores and produces an exponential scaling up to our inflection point where a peer is connected to the majority.

We use the consensus term to scale the original rankings. As peers attain more weight in the network they increase their inflation exponentially up to 0.5. In section 10 we show how this ensures that the larger of two competing sub-graphs comes to own an exponentially larger proportion of the network through inflation.

$$\mathbf{I} = \mathbf{R} \cdot \mathbf{C} \tag{5}$$

3 Bonds

This consensus described above protects against naive collusion by making it difficult for small groups to achieve inflation. However, it does not provide a incentive for correctly selecting weights. We introduce these incentives by adapting the inflation mechanism with a speculation based reward in the form of 'bonds' $\mathbf{B}$. Here, $b_{i,j} \in \mathbf{B}$ is the proportion of bonds owned by peer i in peer j.

$$\mathbf{B} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix} \tag{6}$$

Bonds accumulate at each step similarly to token inflation where $\Delta\mathbf{B} = \mathbf{W} \cdot \mathbf{S}$. In this way, peers accumulate bonds in the peers they rank, thus 'bonding' themselves to those that they are connected to.

$$\mathbf{B}_{t+1} = \mathbf{B}_t + \mathbf{W} \cdot \mathbf{S} \tag{7}$$

Using the $\mathbf{B}$ bond matirx, the chain redistributes the normal incentive scores $\Delta S = \mathbf{B}^T \cdot \mathbf{I}$. Like market based speculation on traditional equities, the peers that have accumulated bonds in peers that others will later value attain increased inflation themselves. Thus it makes sense for peers to accumulate bonds in peers which it expects to do well according to other peers with stake in the system - thus speculating on their future value. Finally, we adapt this mechanism slightly to ensure peers attain a fixed proportion of their personal inflation. For instance, 50 percent, $\Delta S = 0.5\mathbf{B}^T\mathbf{I} + 0.5\mathbf{I}$. ΔS becomes the mechanism step update with determines network incentives across the n peers.

$$\mathbf{S}_{t+1} = \mathbf{S}_t + \tau\Delta S \tag{8}$$

4 Reaching Consensus

The incentive function in section 2 rewards highly trusted peers, however, it may not solve the collusion problem if the honest nodes do not reach consensus. Notably loose, unused stake or incorrectly set weights will detract from the inflation proportion of honest peers in comparison to a colluding sub-network. The honest network, although holding more stake, may not gain enough inflation to overshadow its adversary. The dishonest sub-graph need only attain enough inflation to compete with its largest competitor, not to entirely dominate the network.

This attack is possible when the majority of token inflation is being distributed towards peers which are non-majority-trusted in the graph. The chain can measure this through a 'loss term' $\mathcal{L} = -\mathbf{R} \cdot (C - 0.5)$ (Figure 7). The term is negative iff the majority of inflation is being distributed towards peers with more than 0.5 consensus. The chain uses the loss calculation as a peg. By increasing the number of weights the average miner sets across the network the chain can ensure consensus.

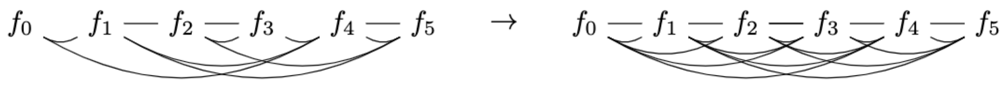


Figure 5: The left network has low consensus $\mathcal{L} > 0$. The system is not resistant to a cabal with less than 50 percent of the stake. The chain increases the number of edges set by peers until $\mathcal{L} < 0$. At this point the majority of inflation flows to peers with majority consensus.

5 Running the Network

The steps to run a peer in the network are:

1. The peer defines its dataset D_i, loss $\mathcal{L}_i$ and parameterized function f_i.
2. At each training iteration, the peer conditionally broadcasts batches of examples from D_i to its peers $x = [\text{batch\_size, sequence\_length, input\_size}]$.
3. The responses $F(x) = [...f_j(x)...]$ – each of common shape $f_j(x) = [\text{batch\_size, sequence\_length, output\_size}]$ – are joined using the gating function and used as input to the local model f_i.
4. Comparison against the target labels produces a loss-gradient $\frac{\partial \mathcal{L}}{\partial F}$ which back-propagates through f_i and out to the network.
5. During 2 and 3 the peers learn the weights for their row $w_{i,j} \in \mathbf{W}$ by measuring the value of the signals produced by their peers.
6. At distinct time-step t participants submit changes to the weights $\Delta \mathbf{W}_i$ to update the ranking $\mathbf{R}$, inflation $\mathbf{I}$, consensus term $\mathbf{C}$, and bond distributions $\delta \mathbf{B}$.
7. The chain measures 'loss' and optionally distributes newly minted stake into the network ΔS according to the bond ownership.

6 Tensor Standardization

A common encoding of inputs and outputs is required for the various model types and input types to interact. The use of tensor modalities can be used to partition the network into disjoint graphs. At the beginning, the network can be seeded with a single modality TEXT, then expanded to include IMAGE, SPEECH, and TENSOR. Eventually, combinations of these modalities can be added; for instance TEXT-IMAGE, to bridge the network into the multi-modality landscape. Incentives to connect modalities can be integrated with the same trust scaling suggested in section (2). Eventually, successful models should accept inputs from any modality and process them into a useful representation. For consistency, we can use a standard output shape across the network $[\text{batch\_size, sequence\_dim, output\_dim}]$ similar to the common tensor-shapes produced by language and image models – and extend this size as the network increase in complexity.

$$\begin{tikzcd}
\text{TENSOR} \arrow[dr, "{[\text{bs, seq, input}]}"] & & \\
\text{IMAGE} \arrow[r, "{[\text{bs, seq, chans, rows, cols}]}"] & f(x) \arrow[r] & {[\text{bs, seq, output}]} \\
\text{TEXT} \arrow[ur, "{[\text{bs, seq}]}"'] & &
\end{tikzcd}$$

Figure 6: Standardization of input dimensions within the network

By working on abstract input classes we can ensure participants work towards a general multi-task understanding Kaiser et al. [2017]. Participants may use: (1) completely distinct computing substrates Nugent and Molter [2014], (2) datasets Lample and Conneau [2019], (3) models, and (4) strategies for maximizing their incentives in the market. It makes sense for peers to work on unsupervised datasets where data is cheap and privacy not required.

7 Conditional Computation

As the network grows, outward bandwidth is likely to become a major bottleneck. The need to reduce network transfer and a method of selecting peers is required. Conditional computation can be used where peers learn through gradient descent how to select and prune neighbors in the network. For example, a product key layer or a sparsely gated layer Shazeer et al. [2017].

$$f_i = f_i(G(x)) \tag{9}$$

$$G(x) = \sum_j g_j(x) * f_j(x) \tag{10}$$

The conditional layer determines a sparse combination of peers to query for each example and multiplicatively re-joins them, cutting outward bandwidth by querying only a small subset of peers for each example. The method can drastically increase outward bandwidth Shazeer et al. [2017] Ryabinin and Gusev [2020], allowing peers to communicate with many more neighbors in the graph. In essence, the layer acts as a trainable DNS lookup for peers based on inputs. Furthermore, being trainable with respect to the loss, it provides a useful proxy for the weights $w_{i,j} \in \mathbf{W}$.

8 Knowledge Extraction

Dependence between functions ensures that models must stay online and cannot be run in production. Breaking this dependence can be achieved using distillationHinton et al. [2015]: A compression and knowledge extraction technique in which a smaller model – the student - mimics the behaviour of the remaining network. The distillation layer is employed in conjunction with conditional computation (10) layer where the student model learns to mimic the network using the cross-entropy (shown below as KL) between the logits produced by the gating network and the student's predicted distribution Sanh et al. [2020].

$$\text{distillation loss} = \text{KL}_D(\text{dist}(x), G(x)) \tag{11}$$

Because the distilled model acts as a proxy for the network, models can be fully taken off-line and evaluated. Recursion through the network is also cut between components allowing for arbitrary network graphs. If models go offline, their peers can use the distilled versions in-place. Private data can be validated over the distilled models instead of querying the network. Eventually, components can fully disconnect from the network using the distilled models to do validation and inference offline.

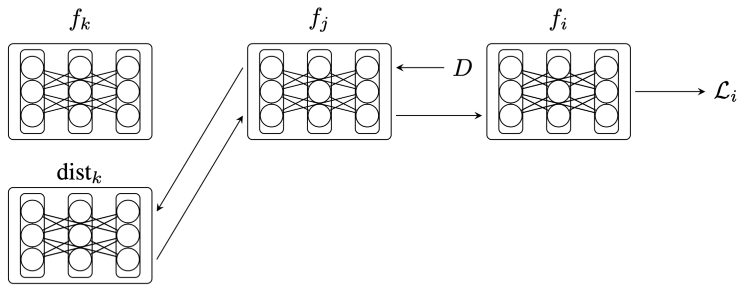


Figure 7: Queries propagate to depth=1 before the distilled model is used.

9 Learning Weights

Our goal in this work is the production of a ranking $\mathbf{r} = [r_i]$ over peers where score $r_i \in \mathbf{R}$ represents a participant's information-theoretic significance to the benchmark. Following LeCun and others LeCun et al. [1989]; Yu et al. [2017], it is reasonable to define this significance by equating it with the cost of removing each peer from the network. We can derive this score analytically where $\Delta F(x)_i$ is a perturbation of the j^{th} peers's inputs when the i^{th} peer is removed from the network (Appendix 12.2):

$$r_i \approx \frac{1}{n} \sum_{j}^{n} \sum_{x \in D_j} \Delta F^T(x)_i \cdot \mathbf{H}(\mathcal{Q}_j(x)) \cdot \Delta F(x)_i \tag{12}$$

$$\Delta F(x)_i = [0, ...0, -f_i(x), 0, ...0]$$

Note, when the error function $\mathcal{Q}_j$ is the twice-differentiable cross-entropy, then $\mathbf{H}(\mathcal{Q}_j)$ is its Fisher-information matrix, and $r_i \in \mathbf{R}$ is suitably measured as each peer's *informational significance* to the network as a whole. However, information theoretic weights require the full Hessian of the error. In practice it is more reasonable to use a heuristic to propagate a contribution score from the error function through to the inputs Yu et al. [2017]. For instance, weights from the gating layer (Section 6) provide a useful differentiable proxy.

10 Collusion

We consider the scenario where a subset of the peers in the network have formed a 'cabal': A set of colluding peers attempting to maximize their inflation without accurately scoring their neighbors. The fight between the honest graph A with stake $\mathbf{S}_A$ and the disjoint cabal B with stake $\mathbf{S}_B$ can be determined by the proportion of network stake held by each. The honest graph must attain more inflation to maintain its dominance and protect the network $I_A >> I_B$.

We assume that the proportion of stake in the honest graph is more than that found in the dishonest graph $S_A > S_B$ and that the chain has reached consensus $\mathcal{L} < 0$. Since all peers in B are disjoint from A, our loss term $-R_B \cdot (C_B - 0.5) > 0$ is positive. Because $\mathcal{L} < 0$ it must be the case that $R_A \cdot (C_A - 0.5) < 0$ is negative and there are peers in the honest sub-graph A who are connected to the majority.

As the chain progresses, newly minted stake is being emitted at our inflation rate τ in proportion to $I = R \cdot T$. Importantly, the gradient of the incentive function with respect to the stake is positive and super-linear at our inflection point between the honest and dishonest graph. Notably, $\frac{\delta I}{\delta S} = \frac{5}{2}$, this ensures that the amount of stake held by each sub-graph reflect a non-linear change in their inflation at the next iteration.

Initially, since $S_A > 0.5$ and $S_B < 0.5$ the proportion of stake emitted in sub-graph A exceeds that in sub-graph B, and sub-graph A's incentive grows super-linearly compared to B. The result is that the ratio of stake $\frac{S_B}{S_A+S_B}$ decreases – the cabal must continually add stake to its sub-graph to maintain itself through time.

We consider this proportion between the competing graphs under continuous inflation. Converting to python code ...

```
tau = 0.1
temp = 10
stake_A = 0.51
stake_B = 0.49
history = []
for block in range(100):
    total_stake = stake_A + stake_B
    trust_A = 1/(1 + math.exp(-(stake_A/total_stake - 0.5) * temp))
    trust_B = 1/(1 + math.exp(-(stake_B/total_stake - 0.5) * temp))
    ranks_A = stake_A
    ranks_B = stake_B
    incentive_A = ranks_A * trust_A
    incentive_B = ranks_B * trust_B
    total_incentive = incentive_A + incentive_B
    total_stake = stake_A + stake_B
    stake_A += tau * total_stake * incentive_A / total_incentive
    stake_B += tau * total_stake * incentive_B / total_incentive
    print (block, stake_B / (stake_A + stake_B))

>>
block | size of cabal
0 0.4877323388820201
1 0.4849535784321247
2 0.4815511535094221
3 0.477389901500398
4 0.4723093486843246
5 0.46612224574620587
6 0.45861590847737577
7 0.44955887540065376
8 0.43871643745912897
9 0.42587900870651624
10 0.41090548935459825
...
90 0.0002827251010618101
91 0.00025719653886131316
92 0.0002339730247373799
93 0.000212846436856568
94 0.00019362744329658293
95 0.0001761438058611274
96 0.00016023883697158944
97 0.00014576999582759936
98 0.00013260761127280887
99 0.00012063371993464691
```

11 Conclusion

We have proposed an intelligence market which runs on a P2P network outside of a trusted environment. Crucially, the benchmark measures performance as representational-knowledge production using other intelligence systems to determine its value. The fact that this can be done in a collaborative and high-resolution manner suggests that the benchmark could provide a better reward mechanism for the field in general.

To achieve this aim, the paper began with the definition of a P2P network composed of abstractly defined intelligence models. We showed how this framework allowed us to produce a ranking for each peer based on the cost to prune it from the network. Peers negotiated this score using a set of weights on a digital ledger. However, the system was incomplete without mechanisms that prevented participants from forming dishonest sub-graphs.

To resolve this, we proposed an incentive scheme based on peer connectivity which exponentially rewarded peers for being trusted by a large portion of the network. This ensured that over time dishonest sub-graphs decay to irrelevance.

Following this, we showed (1) how peers reduced the network bandwidth by learning connectivity using a differential layer and (2) how they could extract fully network-disconnected machine learning models to run in production. The result is an intelligence market which rewards participants for producing knowledge and making it available to new learners in the system.

References

R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green ai," 2019.

OpenAI, "Openai licenses gpt-3 technology to microsoft," *OpenAI Blog*, vol. 1, no. 1, p. 1, 2020.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

F. Chollet, "On the measure of intelligence," 2019.

G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel, "Optimal brain damage." in *NIPs*, vol. 2. Citeseer, 1989, pp. 598–605.

R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," 2017.

L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit, "One model to learn them all," 2017.

M. A. Nugent and T. W. Molter, "Cortical processing with thermodynamic-ram," 2014.

G. Lample and A. Conneau, "Cross-lingual language model pretraining," 2019.

N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," 2017.

M. Ryabinin and A. Gusev, "Towards crowdsourced training of large neural networks using decentralized mixture-of-experts," 2020.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2020.

D. Balduzzi, W. M. Czarnecki, T. W. Anthony, I. M. Gemp, E. Hughes, J. Z. Leibo, G. Piliouras, and T. Graepel, "Smooth markets: A basic mechanism for organizing gradient-based learners," 2020.

P. Dütting, Z. Feng, H. Narasimhan, D. C. Parkes, and S. S. Ravindranath, "Optimal auctions through deep learning," 2017.

12 Appendix

12.1 Ranking Accuracy

We consider the accuracy of the ranking mechanism when peers make self interested updates to the weights on chain. The peers are under incentive pressure to both minimize their loss and stay connected within the largest sub-graph. Since the mechanism is suitably mathematical we model each peer's payoff function as a differentiable utility function of the weights $U(\mathcal{L}(\mathbf{W}))$

$$P_i(\mathbf{W}) = U_i(\mathcal{L}_i(\mathbf{W})) \tag{13}$$

A peer's utility term reflects that peers subjective interest in minimizing their loss $U(\mathcal{L}(\mathbf{W}))$. Since reducing the magnitude of weights connecting this peer to others will decrease its ability to extract knowledge, we can model our utility function as a change in inputs. To model this change we use a shifted threshold i.e. inputs from neighbors are masked when weights drop bellow the average set by other peers $\mu_j = (\frac{1}{n})\sum_i^n s_i * w_{i,j}$ which in turn reflect a change in the loss function of peer i.

$$F_{\mathbf{W}}(x) = [f_0(x) * \sigma(s_i * w_{i,0} - \mu_0), ..., f_n(x) * \sigma(s_i * w_{i,n} - \mu_n)] \tag{14}$$

$$\sigma = \frac{1}{1 + e^{-\frac{x}{T}}} \tag{15}$$

To derive the change in loss given a change in weights we use an input perturbation $(F_{\mathbf{W}} - F_{\mathbf{W}^0})$ where $\mathbf{W}^0$ is the initial choice of weights. The same perturbation equation seen Section 7 returns the change in loss under a Hessian term $\mathbf{H}(\mathcal{L}(F))$ (see 12.3):

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}}\left[(F_{\mathbf{W}} - F_{\mathbf{W}^0})^T \cdot H(\mathcal{L}(F)) \cdot (F_{\mathbf{W}} - F_{\mathbf{W}^0})\right] \tag{16}$$

We make a further linear assumption about the utility function $U_i(W) = \alpha \cdot \mathcal{L}_i(W)$ to give us a fully differential function for a peer's utility. This construction is a smooth market Balduzzi et al. [2020] where we can explore the competitive equilibrium using gradient descent [2] with steps $\Delta \mathbf{W}_i = \frac{\partial P_i}{\partial \mathbf{W}_i}$.

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \lambda \Delta \mathbf{W} \tag{17}$$

$$\Delta \mathbf{W} = \left[\frac{\partial P_0}{\partial \mathbf{w}_0}; \cdots ; \frac{\partial P_n}{\partial \mathbf{w}_n}\right] \tag{18}$$

We evaluate the the accuracy of the peer ranking method by generating statistics from the above empirical model. We first select mechanism parameters $[\lambda, \alpha, n]$ and generate an initial randomized network state $[\mathbf{W}^0, \mathbf{S}]$ with n random positive semi-definite $n \times n$ hessian terms $[\mathbf{H}]$ one for each peer. Given the initialization we apply the descent strategy (18) by computing the gradient terms from (16) and converge the system to the implied equilibrium using a standard gradient descent toolkit. The discovered local minimum is the competitive equilibrium where participants cannot vary from their choice of weights and stand to gain Dütting et al. [2017]. At this point we compute the competitive ranking $\mathbf{R}^*$ and compare it to the idealized score $\mathbf{R}$ derived from the hessians as discussed in Section 7. We measure the difference between the two scores as a spearman-rho correlation and plot example trials bellow.

12.2 Deriving the idealized ranking

We approximate the change in the benchmark $\mathcal{B} = \sum_i^n \mathcal{L}_i$ at a local minimum and under a perturbation $\Delta F(x)_i = [..., -f_i(x), ...]$ reflecting the removal of the i^{th} peer.

$$\Delta \mathcal{B} = \mathcal{B}(F + \Delta F_i) - \mathcal{B}(F) = \sum_i^n \mathcal{L}_i(F + \Delta F_i) - \mathcal{L}_i(F) \tag{19}$$

$$\mathcal{L}_i(F + \Delta F_i) - \mathcal{L}_i(F) \approx \frac{\partial \mathcal{L}_i}{\partial F} \cdot \Delta F_i + \frac{1}{2}\Delta F_i^T \cdot H(\mathcal{L}_i) \cdot \Delta F_i + O(\Delta F_i^3) + O(\Delta F_i^3) \tag{20}$$

[2]Making gradient steps in this game is a regret-free strategy (see 12.6) and achieves the best expected payoff in hindsight.

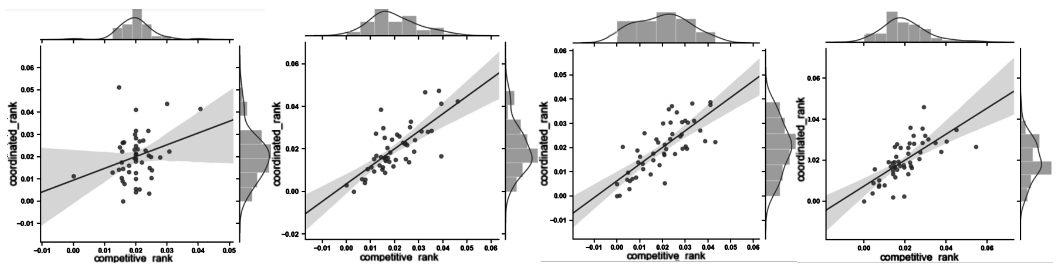


Figure 8: Correlations between the competitive rank and coordinated rank for $\alpha \in \{1, 10, 25, 50\}$. We note that we see an increased relationship between the idealized rank and those discovered by the market improves increasingly through the parameter α.

Equation (19) follows from the definition of the benchmark and Equation (20) follows from a Taylor series under the perturbation $\Delta F(x)_i$. Note that the first term $\frac{\partial \mathcal{L}_i}{\partial F}$ is zero at the local minimum and the higher order term $O(\Delta F_i^3)$ can be ignored for sufficiently small perturbations. These assumptions are also made by LeCun et al. [1989] and Yu et al. [2017]. Note that $\mathcal{L}_i$ is an expectation over the dataset D_i, and all terms are evaluated at a point x so we have:

$$\Delta \mathcal{B} \approx \frac{1}{2} \sum_{i}^{n} \sum_{x \in D_i} \Delta F_i^T(x) \cdot H(\mathcal{Q}_i(x)) \cdot \Delta F_i(x) \tag{21}$$

Here the hessian over the error function $H(\mathcal{Q}_i(x))$ and the summation over the dataset $\sum_{x \in D_i}$ have been appropriately substituted. The constant factor $\frac{1}{2}$ can be removed and this leaves our result.

12.3 Deriving the weight convergence game.

12.4 Theorem

For choice of Hessians $H(\mathcal{L}(F))$ the network convergence-game can be described with the following linear relationship between gradient terms:

$$\frac{\partial P}{\partial \mathbf{W}} = \alpha \cdot \frac{\partial L}{\partial \mathbf{W}} + \frac{\partial r}{\partial \mathbf{W}} \tag{22}$$

With the gradient of the loss:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}} [(F_{\mathbf{W}} - F_{\mathbf{W}_0})^T \cdot H(\mathcal{L}(F)) \cdot (F_{\mathbf{W}} - F_{\mathbf{W}_0})] \tag{23}$$

12.5 Setup

We analyze the system by characterizing the behaviour of participants via their payoff in two terms:

1. The utility attached to that participant's loss as a function of their weights is $U(L(\mathbf{W}))$. U is assumed roughly linear for small change in the weight matrix, $U(\mathcal{L}) = \alpha * \mathcal{L}$, with $\frac{\partial U}{\partial \mathcal{L}} = \alpha$, and α is assumed positive and non-zero.
2. The network is converged to a local minimum in the inputs $\frac{\partial \mathcal{L}}{\partial F} = 0$.

From the payoff formulation in 6.3 we write:

$$P(\mathbf{W}) = \alpha \cdot \mathcal{L}(\mathbf{W}) + r(\mathbf{W}) \tag{24}$$

Note, the utility function and emission were measured in similar units and so α is the *price* of each unit change in loss. The analysis just supposes such a score exists, not that it can be computed. Participants are selecting their weights by making gradient steps $\Delta \mathbf{W}_i = \frac{\partial P_i}{\partial \mathbf{W}_i}$ as to maximize their local payoff. For brevity we omit the subscript i for the remainder of the analysis. Consider a Taylor expansion of the loss under a change ΔF in the inputs.

$$\mathcal{L}(F + \Delta F) = L(F) + \frac{\partial \mathcal{L}}{\partial F}\Delta F + \frac{1}{2}\Delta F \cdot H(\mathcal{L}(F)) \cdot \Delta F + O(\Delta F^3) \tag{25}$$

The first linear term $\frac{\partial L}{\partial F}$ is zero at the assumed minimum and the higher order terms are removed for sufficiently small perturbations in F. We then perform a change of variable $F = F_{\mathbf{W}_0}$, and $\Delta F = F_{\mathbf{W}_1} - F_{\mathbf{W}_0}$ where $\mathbf{W}_0$ are the weights at the minimum and $\mathbf{W}_1$ are another choice such that $F_{\mathbf{W}_0}$ and $F_{\mathbf{W}_1}$ are those inputs masked by $\mathbf{W}_0$ and $\mathbf{W}_1$ accordingly. Substituting this into Equation (25):

$$\mathcal{L}(F_{\mathbf{W}_1}) = L(F_{\mathbf{W}_0}) + \frac{1}{2}(F_{\mathbf{W}_1} - F_{\mathbf{W}_0})^T \cdot H(\mathcal{L}(F)) \cdot (F_{\mathbf{W}_1} - F_{\mathbf{W}_0}) \tag{26}$$

The function $\mathcal{L}(F_{\mathbf{W}_1})$ is simply an approximation of the loss for any choice of weights $\mathbf{W}_1$ given that the network has already converged under $\mathbf{W}_0$. Finally, by the α-linear assumption of the utility we can attain the following:

$$\frac{\partial U}{\partial \mathbf{W}} = \alpha \cdot \frac{\partial L}{\partial \mathbf{W}} \approx \frac{\alpha}{2}\frac{\partial}{\partial \mathbf{W}}[(F_{\mathbf{W}} - F_{W_0})^T \cdot H(\mathcal{L}(F)) \cdot (F_{\mathbf{W}} - F_{W_0})] \tag{27}$$

Note that we've dropped the subscript $\mathbf{W}_1$ for brevity, $L(F_{\mathbf{W}_0}))$ is constant and therefore not depending on the choice of weights, and the fraction $\frac{1}{2}$ can be safely subsumed into the unknown α. The remaining term $\frac{\partial r}{\partial \mathbf{W}}$ is derivable via the ranking function in Section 1. This leaves the result:

$$\frac{\partial P}{\partial \mathbf{W}} \approx \alpha \cdot \frac{\partial L}{\partial \mathbf{W}} + \frac{\partial R}{\partial \mathbf{W}} \tag{28}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}}[(F_{\mathbf{W}} - F_{\mathbf{W}_0})^T \cdot H(\mathcal{L}(F)) \cdot (F_{\mathbf{W}} - F_{\mathbf{W}_0})] \tag{29}$$

12.6 Deriving the ex-post zero-regret step.

Consider the system described above. A set of n peers are changing the weights in the ranking matrix $\mathbf{W}$ iteratively using gradient descent with learning rate λ. $\mathbf{W}^{t+1} = \mathbf{W}^t + \lambda \Delta \mathbf{W}$. Here, the change of weights is $\Delta \mathbf{W} = [\Delta w_0, ..., \Delta w_n]$ where each Δw_i is a change to a single row pushed by peer i. Each peer is attempting to competitively maximize it's payoff as a function of the weights $P_i(\mathbf{W})$.

12.6.1 Definition

The ex-post regret for a single step is the maximum difference in loss between the chosen step Δw_i and all alternative Δw_i^*. The *expected* ex-post regret is this difference in expectation, where the expectation is taken over all choices Δw_j's chosen by other participants Dütting et al. [2017].

$$\text{rgt}_i = E_{\Delta w_j}[\max_{\Delta w_i^*}[P_i(\Delta w_i^*) - P_i(\Delta w_i)]] \tag{30}$$

12.6.2 Theorem

For sufficiently small λ, the expected ex-post regret for strategy $\Delta w_i = \frac{\partial P}{\partial w_i}$ is 0.

12.6.3 Proof

Consider Taylor's theorem at the point $\mathbf{W}$ for the payoff function P under a change in weights $\mathbf{W}^* = \mathbf{W} + \lambda \Delta \mathbf{W}$. There exists a function $h(\mathbf{W}^*)$ such that in the limit as, $\mathbf{W}^* \to \mathbf{W}$ we have the exact equivalence:

$$P(\mathbf{W}^*) = P(\mathbf{W}) + \frac{\partial P}{\partial \mathbf{W}}(\mathbf{W}^* - \mathbf{W}) + h(\mathbf{W}^*) \tag{31}$$

Let $P(\mathbf{W}_*)$ represent the payoff when the weight change of the i^{th} row is $\Delta\mathbf{W}_i = \frac{\partial P}{\partial \mathbf{W}_i}$, and let $P(\mathbf{W}^*)$ be any other choice. Since $\lambda \to 0$, we have $\mathbf{W}^* \to \mathbf{W}$ and by the definition of regret we can write:

$$\text{rgt}_i = E_{\Delta \mathbf{W}_j}[\max_{\Delta \mathbf{W}_i^*}[\frac{\partial P}{\partial \mathbf{W}}(\mathbf{W}^* - \mathbf{W}) - \frac{\partial P}{\partial \mathbf{W}}(\mathbf{W}_* - \mathbf{W})]] \tag{32}$$

This follows by subtracting Equation (31) with choice $\mathbf{W}^*$ and $\mathbf{W}_*$. Next, substituting $\mathbf{W}^* - \mathbf{W} = -\lambda\Delta\mathbf{W}$ and expanding $\frac{\partial P}{\partial \mathbf{W}}\Delta\mathbf{W} = [\frac{\partial P}{\partial \mathbf{W}_0} * \Delta\mathbf{W}_0, ... \frac{\partial P}{\partial \mathbf{W}_n} * \Delta\mathbf{W}_n]$ into the equation above leaves:

$$\begin{aligned} \frac{\partial P}{\partial \mathbf{W}}(\mathbf{W}^{\cdot} - \mathbf{W}) - \frac{\partial P}{\partial \mathbf{W}}(\mathbf{W}_{\cdot} - \mathbf{W}) = \lambda(\frac{\partial P}{\partial \mathbf{W}_i} \cdot \Delta\mathbf{W}_i^* - \frac{\partial P}{\partial \mathbf{W}_i} \cdot \Delta\mathbf{W}_{i*}) + \\ \lambda\sum_{j \neq i}^{n}(\frac{\partial P}{\partial \mathbf{W}_j} \cdot \Delta\mathbf{W}_j^* + \frac{\partial P}{\partial \mathbf{W}_j} \cdot \Delta\mathbf{W}_{j*}) \end{aligned} \tag{33}$$

The constant λ can be removed and the second term depends only on weights of other rows $\mathbf{W}_{j \neq i}$. Since these are independent and evenly distributed these can be removed under the expectation $E_{\Delta\mathbf{W}_j}$. He have:

$$\text{rgt}_i = E_{\Delta \mathbf{W}_j}[\max_{\Delta \mathbf{W}_i^*}[\frac{\partial P}{\partial \mathbf{W}_i} \cdot \Delta\mathbf{W}_i^* - \frac{\partial P}{\partial \mathbf{W}_i} \cdot \Delta\mathbf{W}_{i*}]] \tag{34}$$

Finally, we use the the fact that for vectors a, b and angle between them θ, the magnitude of the dot product is $|a||b|cos\theta$. This is maximized when the vectors are parallel $\theta = 0$ and $cos(\theta) = 1$. In our case, we have the maximum when $\Delta\mathbf{W}_i = \kappa * \frac{\partial P}{\partial W_i}$ for some constant $\kappa > 0$. Thus $P(\Delta\mathbf{W}^*)$ is maximize when $\Delta\mathbf{W}_i^* = \kappa * \frac{\partial P}{\partial \mathbf{W}_i}$. Since $P(\Delta\mathbf{W}^*) = P(\Delta\mathbf{W}_*)$ in the maximum, this proves the point.

12.7 Simulating the Loss Convergence

Bellow is python code for simulating the stake-convergence of a disjoint competing sub-graph against the majority graph.

```
import matplotlib.pyplot as plt
from scipy.sparse import random
from sklearn.preprocessing import normalize
import numpy as np
np.set_printoptions(precision=3, suppress=True)

def random_weight_matrix( n, density ):
    rows = []
    for _ in range( n ):
        row_i = random(1, n, density=density).A # Each row has density_A
        row_i = normalize(row_i, axis=1, norm='l1') # Each row sums to 1.
        rows.append(row_i)
        assert np.shape(row_i)[1] == n
        assert np.isclose(np.sum(row_i), 1.0, 0.001)
    W = np.concatenate(rows)
    assert np.shape(W)[0] == n and np.shape(W)[1] == n
    return W

def random_stake_vector( n ):
    S = np.random.uniform(0, 1, size=(n, 1))
    S = (S / S.sum())
    S = np.reshape(S, [1, n])
    S = np.transpose(S)
    return S

def sigmoid(x):
    return 1 / (1 + np.exp( -x ))

def get_gradient( n, W, R, S, A, C ):
    gradient = []
    for i in range(n):
        grad_i = 0.0
        for k in range(n):
            inner = 0.0
            for j in range(n):
                inner += A[j] * S[j] * C[j, k]
            grad_i += C[i, k] * R[k] * np.exp(inner) * sigmoid(inner) * sigmoid(inner) + W[i, k] * (sigmoid
        gradient.append(grad_i)

    return np.concatenate(gradient)


def cabal_decay(
        nA:int = 5,
        nB:int = 100,
        n_blocks:int = 2000,
        tau:float = 0.01,
        learning_rate:float = 0.05
    ):
    r""" Measures the proportion of stake held by a disjoint sub-graph 'cabal' as a function of steps.

        Args:
            nA (int):
                Number of peers in the honest graph
            nB (int):
                Number of peers in the disjoint graph
            n_blocks (int):
                Number of blocks to run.
            temperature (float):
                temperature of the sigmoid activation function.
            tau (float):
                stake inflation rate.
            learning_rate (float):
                scaling term learning rate.

          Returns:
              history (list(float)):
                  Dishonest graph proportion.
    """

    n = nA + nB
    # Randomized Matrix of weights. We create the subgraphs by concatenating two disjoint
    # and random square matrices.
    W = np.concatenate((
        np.concatenate((random_weight_matrix(nA, 0.9), np.zeros((nA,nB)) ), axis=1),
        np.concatenate((np.zeros((nB,nA)), random_weight_matrix(nB, 0.9)), axis=1)),
        axis=0) ; print ('W \n', W)

    # Randomized Vector of stake.
    # Subgraph A gets 0.51 distributed across values 1->n_a.
    # Subgraph B gets 0.49 distributed across values 1->n_b
    S = np.concatenate((
        random_stake_vector(nA)*0.51,
        random_stake_vector(nB)*0.49),
        axis=0) ; print ('S \n', np.transpose(S))

    # Scaling vector. A is multiplied by S before each iteration
    # to attain our scaled stake.
    A = np.ones(n) ; print ('A \n', A)

    # Matrix of connectivity. We use the true absorbing markov chain calculation
    # In practice this is too difficult to compute and opt for a depth d cut off.
    Wdiag = np.zeros((n,n))
    Wdiag[np.diag_indices_from(W)] = np.diag(W)
    Q = W - Wdiag
    C = np.maximum(np.linalg.pinv(np.identity(n) - Q), np.zeros((n,n)))
    C = np.where(C > 0, 1, -1) ; print ('C \n', C)

    # Iterate through blocks
    history = []
    for block in range(n_blocks):

        # Scale the stake against our vector A.
        S_scaled = S * np.reshape(A, (n,1))#; print ('S * A \n', S_scaled)

        # Compute the ranks W^t S.
        R = np.matmul(np.transpose(W), S_scaled) / np.sum(S_scaled) #; print ('R \n', R)

        # Compute our trust scores sigmoid( (C^T S) * temperature )
        T = 1 /( 1 + np.exp(-np.matmul(np.transpose(C), S_scaled))) #; print ('T \n', T)

        # Compute our chain loss.
        loss = -np.dot(np.transpose(R), (T - 0.5) ) # ; print ('loss \n', loss[0][0])

        # Our loss is negative, so we emit more stake.
        if loss < 0.0:

            # Compute our incentive function.
            I = R * T

            # Distribute more stake.
            S = S + tau * (I/np.sum(I))

            # Measure the size of our cabal.
            S_Honest = np.sum(S[:nA])
            S_Cabal = np.sum(S[nB:])
            ratio = S_Cabal / (S_Honest + S_Cabal)
            history.append(ratio)


        # Our loss is positive, so we update our scaling terms.
        else:
            # Compute the gradient of our scaling terms.
            grad = get_gradient( n, W, R, S, A, C ) #; print ('Grad \n', grad)

            # Update our scaling terms, being sure not to have negative values.
            A = np.minimum(np.maximum( A + learning_rate * grad, np.zeros_like(A) ), 1)


    plt.plot(history)

cabal_decay()
```