BitTensor: A Peer-to-Peer Intelligence Benchmark

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

The dominant tools used to guide machine intelligence are typically datasets of labeled examples which rank systems on a set of predefined tasks. However, we argue that this method based on benchmarking tasks is an inefficient guide for machine intelligence. Instead, we propose a benchmark that ranks *knowledge production* from within a network of intelligence systems. The result is a system that is more efficient, collaborative, and generalized. In this paper we describe how this benchmark is constructed and how it is negotiated by computers that share knowledge peer-to-peer (P2P) across the internet. Since this internet based negotiation introduces game-theoretic considerations, we design methods to ensure the veracity of the benchmark when peers remain self-interested and trustless. We then test our design by empirically showing that the negotiation closely matches the desired ranking at the system's competitive equilibrium.

Increasingly, state-of-the-art machine learning systems focus on the production of *machine knowledge*, non-task specific understanding of inputs, which can be later tuned to a wide variety of problems [1]. This shift is a consequence of the fact that a focus on tasks during training produces narrow specialists rather than resilient generalists [2]. In contrast, while becoming the dominant technique for training individual models, the field still opts for task based objectives as the method of reward at the highest level of its evolution [3]. The misalignment between this salient feature (machine knowledge) and its feedback mechanism (task performance) is inefficient for the following reasons:

1. A definition of intelligence provided by task performance produces a feedback mechanism which converges the field towards a narrow definition. Unfortunately, simply expanding the set tasks to cover the general problem would require a near infinite number of supervised (dataset, task) pairs [2]. Consequently, task-based feedback mechanisms at the highest level of the field make it intractable to achieve a high level of generalism [4].
2. Representational knowledge can be measured in a much higher dimensional space than the low-dimensional scores produced by the tasks. For instance, the 1024 dimensional floating point embedding produced by BERT [1] in comparison to the 8 dimensional score produced by SuperGLUE [5]. In the low dimensional space, we cannot easily divide and conquer the problem to reward niche machine learning systems for working on specific aspects of understanding. This is wasteful as those models may be costly to create in the first place.
3. We do not measure how much one team's model improved the collective. Task based benchmarks produce a winner-take-all competition which prunes all but the best. The result is a world-wide challenge where only those with the means and the resources to train the largest models may contribute to the state-of-the-art.

A benchmark that directly measures machine knowledge production would be very beneficial [1]. By analogy, similar to rewarding domain specialists or generalists in human cultures – not against

[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

their ability to solve tasks – but their ability to improve overall understanding. Our proposal is a framework in which models directly share knowledge that they have learned with each other. Their peers select that knowledge for its ability to improve their own understanding. Crucially, each model's performance is measured by its ability to improve collective understanding of inputs rather than against any specific task. This re-framing of the measurement technique allows the system to reward models for improving small regions of the general problem. Inherently, the benchmark introduces knowledge sharing, while the demand for performance guides a de-facto "market for intelligence".

Practically, we design the benchmark to run in a continuous and asynchronous fashion, peer-to-peer (P2P) across the Internet. Since this introduces trustless computation [2] we dedicate a part of this paper to explaining how the system remains fair when little assurance can be given about the computers that compose it. In summary, the contributions of this paper are thus four-fold:

1. We introduce a P2P based intelligence framework.
2. We introduce an "intelligence market" that rewards models that improve representational knowledge within the network.
3. We show how models can share knowledge to increase each other's performance, which removes the need for others to re-learn that knowledge.
4. We show how a self-interested desire to minimize a loss function can be used to guide the network towards a fair ranking.

The rest of this paper is organized as follows: Section 1 presents the intelligence benchmark and ranking mechanism that models use to evaluate each other. Section 2 proposes a method to detect and respond to the scenario where participants are not honestly reporting the significance of their peers. Section 3 highlights the experiments performed to verify the function of this framework. Finally, Section 4 presents a summary and future works on this system.

1 Methodology

1.1 Benchmark

The network is composed by n unique parameterized functions $F = f_0, ..., f_j, ...f_n$ where each function is producing an output tensor $f_i(F(x))$, a "representation" from an input tensor $F(x) = [f_0(x)...f_n(x)]$ gathered by querying its neighbors. Each function is training asynchronously over a dataset $D_i = [X, Y]$ such that, given an error function $\mathcal{Q}_i$, its expectation over that data E_{Di} defines a loss $\mathcal{L}_i = E_{Di}[\mathcal{Q}_i(\ y,\ f_i(F(x))\ )]$. We assume these losses are measured on the same scale and thus our benchmark $\mathcal{B}$ can be defined by their sum:

$$\mathcal{B} = \sum_{i}^{n} \mathcal{L}_i \tag{1}$$

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

Figure 1: $n = 6$ parameterized functions with losses $\mathcal{L}_i$ and datasets D_i.

Each parameterized function is represented here in its most abstract sense[6] and need only accept the same input type x and produce the same output dimension to fit within the network. For instance, unicode encoded string $x = ”hello”$ and its semantic representation as a $64x49$ word embedding. This widened scope ensures participants can be multi-task [7], use completely distinct computing substrates [8] or train on unique datasets. [9].

[2] In decentralized systems, trust is shifted from the individuals to the network itself, "trustless" usually means "minimal trust is required" with respect to the protocol

1.2 Ideal Ranking

Our goal in this work is to produce a ranking $R = [R_i]$ over these functions where the score $R_i \in R$ represents participant i's information-theoretic significance to the benchmark $\mathcal{B}$. Following Le Cun and others [10, 11], it is reasonable to analytically define this significance by equating it with the cost of removing each component from the network:

$$R_i \approx \frac{1}{n}\sum_{j}^{n}\sum_{x\in D_j} \Delta F^T(x)_i * H(\mathcal{Q}_j(x)) * \Delta F(x)_i \tag{2}$$

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

Where the above is derived using a Taylor series (Appendix 6.1) and $\Delta F(x)_i$ is the perturbation of the i^{th} node's inputs as a function of it's choice of weights when removing function f_i at the point x. Note, the linear and higher order terms of the Taylor series have been removed following [11] and the remaining term $H(\mathcal{Q}_i)$ is the hessian of our error function. When the error function $\mathcal{Q}$ is the twice-differentiable cross-entropy, then $H(\mathcal{Q}_i)$ is the Fisher information matrix, and $R_i \in R$ is measured as relative entropy: reflects each participants informational significance to the network as a whole.

1.3 Inter Ranking

It is not possible to compute the ranking score above without access to the parameters of each function in the network. Instead, we use a set of inter-model weights $W = [w_{i_j}]$ where each $w_{i,j}$ is the score attributed to f_j from f_i combined into an $n \times n$ square matrix.

$$w_{ij} = \sum_{x\in D_i} \Delta F^T(x)_j * H(\mathcal{Q}_i(x)) * \Delta F(x)_j \tag{3}$$

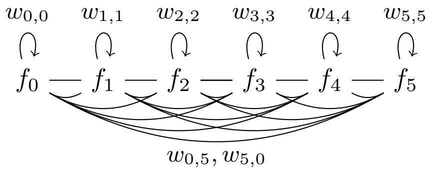

Figure 2: Inter-model contribution weights: $w_{i,j}$ the score attributed to f_j from f_i

The weights can be computed on the fly either approximately [11] or by using the full hessian of the error. We store them on a distributed ledger and allow participants to update them by making changes of bounded size: $W^{t+1} = W^t + \lambda\Delta W$, where $||W_i||_2 < \epsilon$ at block step t. We also enforce that the scores in each row sum to 1, $||w||_1 = 1$. The reasoning for this enforcement is to ensure stake fairness between early movers and newcomers with innovations that enable them to perform better overall.

The equivalent ranking R in Equation (2) can then be computed by normalizing column sum of the weight matrix:

$$R = \frac{1}{n}W^T * \mathbb{1} \tag{4}$$

The problem is that without system-wide access to the model parameters, the computation of w_{ij} in Equation (3) is intractable, as it would require collecting great amounts of information from potentially billions of neurons. It is reasonable to assume participants will select weights which artificially increase their own rank rather than others in the network. Moreover, since the network remains open, participants may choose to create many spuriously neighbours and rank themselves higher. The remainder of this paper describes our proposal for resolving these issues.

1.4 Stake

The proposed solution begins by introducing a finite resource $S = [s_i]$, a component's 'stake' in the system, and an inflation mechanism τ which translates the ranking vector R into additional stake as incentive.

$$R = \frac{1}{n} W^T \circ S * \mathbb{1} \tag{5}$$

$$S^{t+1} = S^t + \tau * \frac{R}{||R||_2} \tag{6}$$

The $\circ$ represents Hadamard product between the $n \times n$ matrix W and the $n \times n$ matrix containing S in each column, and t is the time-step referred to in Section 1.3 (measured in distinct blocks on the distributed ledger). By design Equation (5) increases the importance of those with stake, $s_0 * w_{ij}$. This serves two purposes:

1. New computers can spuriously create new nodes, however this won't game the ranking because the amount of stake they hold is finite.
2. The resource provides mechanism power.

By providing it to nodes with large rank, this ensures that those with weight must have worked to attain it, or indirectly subsidized those who have done so already. A single staked token would be enough to bootstrap the process. [3]

Algorithm 1 Inflation mechanism

```
Require: $S = [n \times 1]$ Stake Vector
Require: $W = [n \times n]$ Weight Matrix
Require: $\tau > 0$ inflation rate
  while TRUE do
    $W = W + \lambda \Delta W$
    $R = \frac{1}{n} W^T \circ S * \mathbb{1}$
    $S = S + \tau * \frac{R}{||R||_2}$
  end while
```

1.5 Competitive weights

While stake provides some protection against malicious actors, it does not ensure weights are set accurately. Our solution begins by introducing competition for connectivity within the network. Nodes that underweight are punished by having inputs from the network masked to zero (7), which in turn would punish the loss function. To frame this market we borrow the continuous differential activation function σ with range $(0, 1)$. Under a choice of weights W_i the inputs to component i are:

$$F_W(x) = [f_0(x) * \sigma(s_i * w_{i,0} - \mu_0), ..., f_n(x) * \sigma(s_i * w_{i,n} - \mu_n)] \tag{7}$$

$$\sigma = \frac{1}{1 + e^{-\frac{x}{T}}} \tag{8}$$

Here, the shift term μ_j is the average of the weights in each column $\mu_j = (\frac{1}{n}) \sum_i^n s_i * w_{i,j}$, and the activation function is the temperature scaled sigmoid. Because the allocation mechanism is standard across the network it is possible for each participants to compute both $\frac{\partial \mathcal{L}_i}{\partial W_i}$ and $\frac{\partial R_i}{\partial W_i}$. Computers may augment their usual training framework, for instance, Tensorflow, with the allocation mechanism shown here.

[3] The propagation of a resource mocks the flow of the brain-derived neurotrophic factor (BDNF). [12]

1.6 Running the network

The steps to run a network participant are:

1. Participant defines its dataset D_i, loss $\mathcal{L}_i$ and parameterized function f_i
2. At each training iteration the participant broadcasts batches of examples from D_i to its peers $[batch\_size, x]$
3. Responses $F(x)$ from the network produce a loss-gradient $\frac{\partial \mathcal{L}}{\partial F}$ which back-propagates through f_i and out to the network.
4. During 2 and 3 the participant competitively selects the weights for their row $w_{ij} \in W$.
5. Participants submit changes to the weights ΔW_i which, in-turn changes the ranking and induces inflation $\tau * R$.
6. After steps 1-5 have been performed multiple times, participants disconnect and verify the model in a normal manner.

Peers only communicate with computers that hold stake as a consequence of section 1.5. Those that fail to produce value will be pruned naturally as participants learn to differentiate signal from noise.

1.7 Conditional computation

As the network grows, outward bandwidth will become the major bottleneck. Components learn to trim outward bandwidth by employing a Sparsely-Gated Mixture-of-Experts (SGMoE) [13] layer at the input. The gating layer determines a sparse combination of children to query for each example and then re-joins them using the the gating weights $g_j(x)$. The combined gated inputs are fed as input to the local function:

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

$$G(x) = [..., g_j(x) * f_j(x), ...] \tag{10}$$

The layer cuts outward bandwidth, querying only a small subset of peers for each example. The gating function is trainable w.r.t to the loss and its weights act as a proxy for importance $w_{ij} \in W$. This method has been shown to drastically increase the potential for outward bandwidth in datacenter training,[13] and has been investigated in a peer-to-peer (P2P) setting as well [14]

1.8 Extracting knowledge

Inter-node dependence in the network is broken using distillation[6], a compression and knowledge technique in which a smaller model – the student - mimics the behaviour of an ensemble. We employ this technique over the gating ensemble described in Equation (10) where the student model learns to minimize the cross-entropy (shown below as KL) between the logits produced by the gating network and its predicted distribution. [15]

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

We use the distilled model as proxy to cut recursive calling between each components rather than query farther into the network. If models go offline, their peers can use their 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 inputs to validate and inference the models offline.

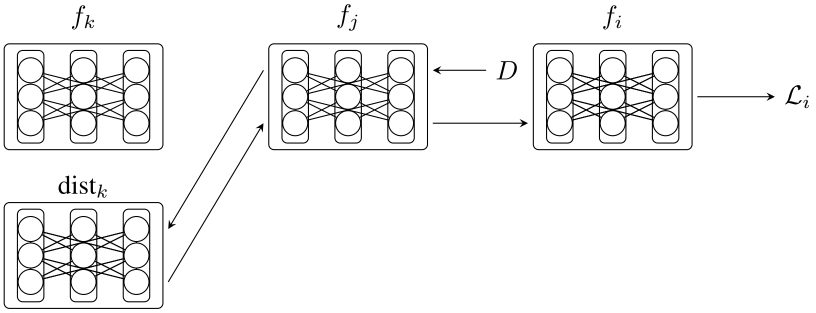

Figure 3: Queries propagate to depth=1 before the distilled model is used.

2 Analysis

We consider the scenario where participants are not honestly reporting the significance of their peers. The network is progressing at discrete timesteps t and it is reasonable to assume that each participant is attempting to maximize their subjective payoff over these steps.

2.1 Payoff Model

The staking system in Section 1.4 gives participants incentive to maximize their self-weight w_{ii} while the competitive connectivity described in Section 1.5 makes it costly for participants to decrease the remaining weights w_{ij} in their row. Since the row must sum to 1, we have a trade-off in two terms:

1. A utility term attached to the loss $U(\mathcal{L}(W))$.
2. The token emission via inflation $\tau * R_i(W)$.

Both of these are functions of the weights and, without loss of generality, measured in similar units:

$$P_i(W) = U_i(\mathcal{L}_i(W)) + \tau * R(W)_i \tag{12}$$

It is reasonable to assume payoff maximizing participants will use P_i as their objective during training. This can be computed using standard tools since both terms $U(\mathcal{L}_i(W))$ and $R(W)_i$ are fully continuous and differentiable. The system can be characterized as a competitive gradient descent game where participants are making steps $\Delta W_i = \frac{\partial P_i}{\partial W_i}$. In appendix 6.2 we prove that this strategy is regret-free and achieves the best expected payoff in hindsight. This assumption is also generally employed in smooth markets [16]. The iterative descent thus follows:

$$W^{t+1} = W^t + \lambda \Delta W \tag{13}$$

$$\Delta W = [\frac{\partial P_0}{\partial W_0}; ...; \frac{\partial P_n}{\partial W_n}] \tag{14}$$

2.2 Empirical Model

Without access to the running network we evaluate our system using an empirical model. To derive the gradient steps in this model $\frac{\partial P_i}{\partial W_i}$ we make the following assumptions:

1. The utility functions are continuous-differentiable and are bounded by their first order derivatives $\frac{\partial U}{\partial \mathcal{L}} = \alpha$.
2. The network is converged to a local minimum in the inputs $\frac{\partial \mathcal{L}}{\partial F} = 0$.

The first assumption is approximate for small changes of continuous functions and the second assumption is realistic after extended training. In Appendix 6.2 we derive the following gradient step:

$$\frac{\partial P}{\partial W} = \frac{\alpha}{\tau} * \frac{\partial L}{\partial W} + \frac{\partial R}{\partial W} \tag{15}$$

$$\frac{\partial L}{\partial W} = \frac{\partial}{\partial W}[(F_W - F_{W_0})^T * H(\mathcal{L}(F)) * (F_W - F_{W_0})] \tag{16}$$

Here F_W is the masked inputs from Section 1.5. $(F_W - F_{W_0})$ is the difference in the mask between the choice of weights W and the weights at the minimum W_0 and $H(\mathcal{L}(F))$ is the $[n \times n]$ hessian of the loss over inputs F. This formulation is both intuitive and useful: the gradient term $\frac{\partial L}{\partial W}$ measures the change in loss for any choice of weights. Additionally, it also allows us to compute the ranking for simulated hessian terms.

The remainder of the network is deterministic, and can be described by a choice of $\theta = [\alpha, \tau, \lambda, n, \sigma, T, S, W, H]$. For instance, the secondary term, $\frac{\partial R}{\partial W}$ can be computed-directly from Section 1.4 and only depends on the stake vector S and weights W.

3 Experiments

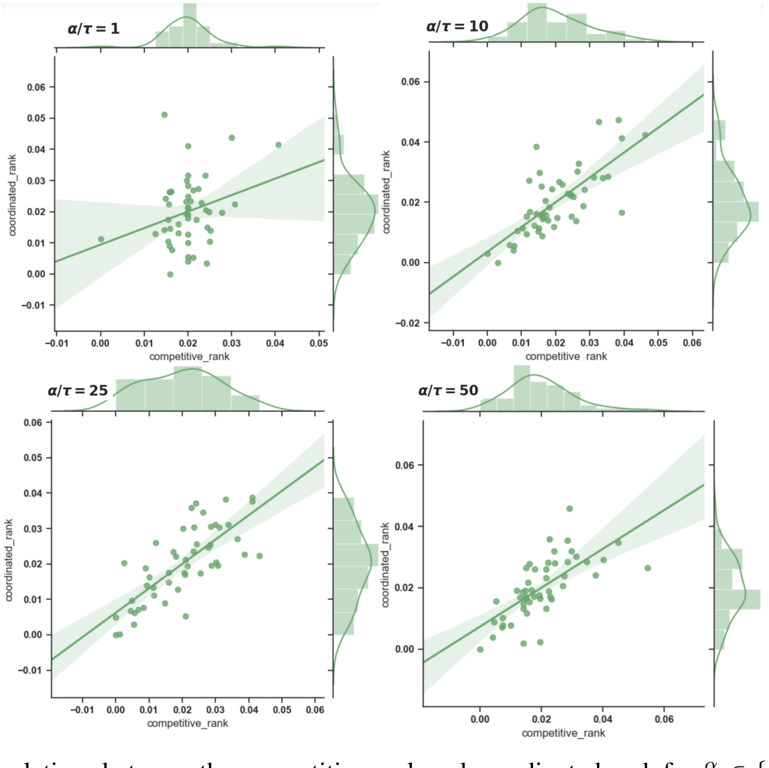

Figure 4: Correlations between the competitive rank and coordinated rank for $\frac{\alpha}{\tau} \in \{1, 10, 25, 50\}$. For low values of $\frac{\alpha}{\tau}$ the weights converge to the identity: the state where peers are fully disconnected.

To generate sample statistics from the network, we first select $[\alpha, \tau, \lambda, S, \sigma, T, n] \in \theta$, then we generate random positive semi-definite hessians H, and random uniform initial weights W_0. For each parameterization we discover the competitive ranking by converging the system to a Nash-equilibrium: an equilibrium where no individual can vary from their set of weights and stand to gain [17]. To find this equilibrium, we use the competitive descent strategy described in (16) and compute the gradient terms from (14). In each trial we use a learning rate $\lambda = 0.005$ and stop when the gradient terms are bounded by ϵ or the steps exceed $1000 \times n$. The competitive ranking R^* at this point follows from (4) and can be compared to the idealized score in R from (2).

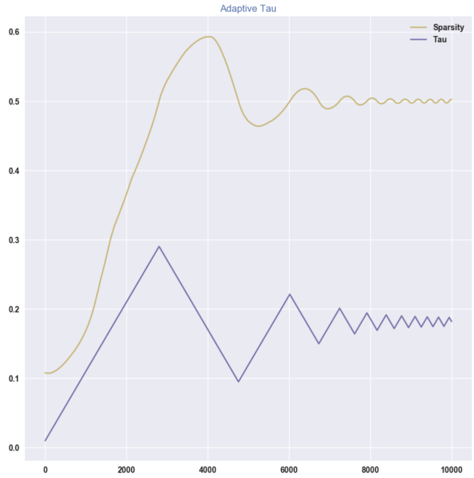

Figure 5: (i) Left: the ratio between the main diagonal and the remainder of the weights. Right: the adaptive τ parameter converging onto the target. The weight matrix sparsity is a proxy for the ranking accuracy which we see in Figure 4. As sparsity converges onto the target the ranking is also improving.

We are interested in the correlation between R^* and R as we vary the ratio α and τ, this ratio is explicit in (15) where the fraction relates the two gradient terms. Intuitively the ratio is between the value of minimizing the loss and maximizing revenue from inflation. Since this effects the ranking, we show this trade-off for various choices in Figure 4.

Finally, in Figure-5 we implement an adaptive-τ strategy where the network varies the inflation rate. Initial inflation is zero and then increases until the weights begins to converge towards the main diagonal $w_{ii} = 1$. We measure the sparsity

$$sparsity = \frac{\sum W_{dg}}{\sum W} \tag{17}$$

as the ratio between the main diagonal and the remaining weights. As sparsity increases we push the market equilibrium by decreasing τ. Figure 5 shows this adaptive convergence for $\alpha = 1$ with a sparsity target of 1.

3.1 Discussion

Figure 4 shows the relationship between the idealized rank and the competitive rank as a function of $\frac{\alpha}{\tau}$. Figure-4-a, $\frac{\alpha}{\tau} = 1$ shows the case where this ratio negatively effects the ranking accuracy. Here, all components have set $w_{ii} = 1$ and the resulting scores for all participants has converged to $1/n$. When this occurs, the system could decrease the inflation rate τ and push the network towards the high information markets seen in Figure-4-b, c, and d. Figure-5 shows a basic implementation of this where τ adapts to the ratio between the row-sums and the main-diagonal. By lowering inflation, it subsequently 'costs less' to connect with peers. Profit maximizing nodes automatically adjust to the change and the system converges back towards an accurate ranking. Those with high ranks will oppose inflation decreased, while those with low ranks will welcome it. The equilibrium found in this meta game will most certainly depend on the number of participants, key to both the ranking accuracy and the market at its core. However, we leave this analysis for a follow up paper.

4 Conclusion

We have proposed an inter-model benchmark that can run in a P2P setting outside of a trusted environment. We started with a typical machine learning framework defined by a set of functions with their losses over given datasets, then derived an idealized ranking score. The measure produced an information theoretic score that new participants can improve by learning how to be useful to their peers. However, the system is incomplete without a mechanism that prevents participants from ranking dishonestly. To resolve this, we proposed an incentive scheme and a differential allocation system over the network weights. The system allows the participants to train for connectivity in the graph. Following this, we described how to increase the outward bandwidth in the system using a trainable gating network and how to cut dependence between nodes using distillation. Finally we showed how increasing the number of nodes in the system and fixing the inflation mechanism properly ensured that the resulting rank scores would correlate with those found in a idealized setting. While this is true, stake in the system holds value as a means to drive what the network learns. That benchmark is continually being solved by the participants, compounding what has been learned before and making it available to new learners in the system.

5 Broader Impact

In order to provide a balanced perspective, authors are required to include a statement of the potential broader impact of their work, including its ethical aspects and future societal consequences. Authors should take care to discuss both positive and negative outcomes.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[2] 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.

[3] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, “Glue: A multi-task benchmark and analysis platform for natural language understanding,” *arXiv preprint arXiv:1804.07461*, 2018.

[4] F. Chollet, “On the measure of intelligence,” *arXiv preprint arXiv:/1911.01547*, 2019.

[5] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, “Superglue: A stickier benchmark for general-purpose language understanding systems,” in *Advances in Neural Information Processing Systems*, 2019, pp. 3261–3275.

[6] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015.

[7] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit, “One model to learn them all,” 2017.

[8] M. A. Nugent and T. W. Molter, “Cortical processing with thermodynamic-ram,” 2014.

[9] G. Lample and A. Conneau, “Cross-lingual language model pretraining,” 2019.

[10] Y. LeCun, D. J. S, and S. S. A, “Optimal brain damage,” *Advances in Neural Information Processing Systems 2 (NIPS)*, 1989.

[11] 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.

[12] B. S and D. UN, “Brain-derived neurotrophic factor and its clinical implications,” *Arch Med Sci. 2015;11(6):1164–1178. doi:10.5114/aoms.2015.56342*, 2015.

[13] 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.

[14] M. Riabinin and A. Gusev, “Learning@home: Crowdsourced training of large neural networks using decentralized mixture-of-experts,” *arXiv preprint arXiv:/2002.04013*, 2020.

[15] L. C. J. W. T. Sanh, Victor; Debut, “Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter,” *arXiv preprint arXiv:1910.01108*, 2019.

[16] 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.

[17] P. Dütting, Z. Feng, H. Narasimhan, D. C. Parkes, and S. S. Ravindranath, “Optimal auctions through deep learning,” 2017.