Training Deep Neural Nets to Aggregate Crowdsourced Responses

Alex Gaunt
Microsoft Research
Cambridge, UK

Diana Borsa
University College London,
London, UK

Yoram Bachrach
Microsoft Research
Cambridge, UK

Abstract

We propose a new method for aggregating crowdsourced responses, based on a deep neural network. Once trained, the aggregator network gets as input the responses of multiple participants to the same set of questions, and outputs its prediction for the correct response to each question. We empirically evaluate our approach on a dataset of responses to a standard IQ questionnaire, and show it outperforms existing state-of-the-art methods.

1 INTRODUCTION

Crowdsourcing platforms such as Amazon's Mechanical Turk are marketplaces which bring together participants who wish to take part in small micro-tasks in return for a fee and businesses or individuals who want to employ people for such tasks. A common use-case is data annotation and labelling, such as determining whether a caption correctly describes an image, or deciding whether the sentiment expressed in a text is positive, neutral or negative. Annotators may not always produce the correct responses to the questions posed to them. This might happen for several reasons: they may lack the sufficient knowledge to accurately answer every question or they might not be exerting the required effort to do well on a task. As annotators are not completely reliable in producing the correct responses to the posed questions, the resulting set of annotations would include errors, which results in an *inherent uncertainty* regarding which answers are correct. One instrument that can reduce the number of such errors is using *redundancy*: rather than asking a single participant to answer a question, the same question is posed to multiple participants; the multiple responses to the same question can then be *aggregated* into a single response to the question.

A commonly used aggregator is "majority vote", in which the chosen answer to each question is the one given most frequently by the participants. [1] This form of collective decision making has been heavily investigated. A result from the 18th century by the Marquis de Condorcet [5] states that the probability of majority vote to reach the correct conclusion to a question with two possible answers approaches 1 as the number of aggregated participants approaches infinity, assuming that participants are independent and each has a probability $p > \frac{1}{2}$ to provide the correct answer. [2]

Using crowdsourcing marketplaces it is possible to gather the responses of many participants to many questions in a very short time frame. Thus, such services make it easy to harness the collective intelligence of many participants. Given a set of responses of multiple participants to the same questions, one can use majority vote to get the aggregate responses to all the questions. However, this simple aggregator may be suboptimal, as some annotators produce higher quality results than others. For instance, if we know that the annotator Alice is more reliable than Bob, we might give her responses more weight when computing the aggregate solution. One way to determine the ability of annotators to provide correct responses is to evaluate each participant on a "gold-set" of questions, for which the correct answer is known. Given information about the ability of annotators evaluated on the gold-set, one can better aggregate the responses to a set of questions for which the correct answer is not known. However, what can be done to better aggregate responses in the absence of such a gold-set?

Earlier work has uncovered aggregation algorithms that outperform the majority vote aggregator, and that do not use external information such as a gold-set. Many such techniques are based on Bayesian models, that jointly infer information about the relative abilities and biases of participants, the difficulty levels of questions and the correct answer to each question [23, 2, 21]. As these techniques are Bayesian, they rely on a statistical model of the process through which the data was generated, reflecting model-

[1] The field of social choice refers to this aggregator as "plurality voting".

[2] The majority aggregator has also been shown to perform well in practice in multiple domains [22, 1, 10].

ing assumptions regarding the domain. Such assumptions are captured as random variables and the relation between them, consisting of the observed data and a set of assumed latent (unobserved) variables, along with a joint probability model tying the variables. While such techniques have been shown to be powerful tools for aggregating crowdsourced responses, their performance can be hindered when the modeling assumptions are incorrect or inaccurate – especially in the absence of large amounts of data.

Our contribution:

We propose a novel approach for aggregating crowdsourced responses. As opposed to existing Bayesian algorithms, at the heart of our approach lies a deep neural network, rather than a carefully engineered statistical model encoding an assumed relation between observed variables and possible latent factors. The network is trained by taking a small **seed dataset** of responses of participants to questions to which the correct answers are known. However, as opposed to the gold-set approach, we *only use the seed dataset to train the aggregator network*. Once the network is trained, we evaluate its ability to aggregate responses on a *completely separate* dataset, of both participants and questions the aggregator *has not encountered in the past*. Our training procedure involves using the small seed dataset to synthetically create a large training dataset, reflecting certain desiderata for aggregator functions.

We empirically evaluate our approach on a dataset of responses to a standard IQ questionnaire, and show it has a superior performance over existing methods. We examine the relation between the quantity of data available to our aggregator and its performance. Finally, we explore ways in which the network can infer not only the correct answers but also who the strong annotators are.

2 RELATED WORK

Earlier work in machine learning and artificial intelligence examined methods for merging the opinions of multiple people or agents, covering various aspects such as prediction markets [12] for predicting a future random event, aggregation of information in semantic web platforms [9], hybrid probabilistic relational frameworks [7] which combine logic based representations and probabilistic inference, and information aggregation in peer assessment systems [13].

A domain of particular interest is collective decision making and voting over candidate alternatives. Social choice theory focused on a set of rational agents, each of which has a preference order over the same set of candidates, and examined voting schemes which aggregate these preferences into a single aggregate decision [20] (such as who is the chosen candidate who wins the elections). Social choice theory shows how voting mechanisms allow reaching good group decisions [11], but has also uncovered ways in which voting rules are susceptible to manipulations by voters, who may lie about their true preferences so that the voting rule chooses an alternative their prefer [8]. We assume that the participants' responses reflect their true opinion and focus on the inference problem.

Our work focuses on aggregating crowdsourced opinions. A recent survey examines the implications of label noise in classification [6], and discusses label noise cleaning methods. One approach proposed for this problem is using EM [24], and another approach relies on modeling task difficulty [17]. Further, some algebraic bounds were provided for the binary rating case [4]. Other aggregation methods rely on probabilistic graphical models [23, 14, 18] including the state-of-the-art method of Bachrach et al. [2] (our empirical analysis shows we outperform this method).

3 PRELIMINARIES

We consider a set Q of $|Q| = q$ multiple choice questions, where each question has several possible answers. For simplicity, we denote the possible answers for each question as $[a] = \{1, 2, 3, \ldots, a\}$. For each question $j \in Q$, there is exactly one correct answer $g_j \in [a]$ and we denote the set of correct answers to all the questions as the vector $g = (g_1, g_2, \ldots, g_q) \in [a]^q$. The questions are posed to a set P of $|P| = p$ participants. Each participant $i \in P$ selects a response to each question j, reflecting which of the a possible answers they believe to be the correct one. We denote the response of participant $i \in P$ to question $j \in Q$ as $r_{i,j} \in [a]$. We collect the responses of all participants to all questions in a response matrix $M \in M_{p\times q}$ (where $M_{p\times q}$ denotes the set of all matrices with p rows and q columns) as follows: the element in the i'th row and j'th column in the matrix is the response of participant $i \in P$ to question $j \in Q$ (i.e. $M_{i,j} = r_{i,j} \in [a]$). Thus, each row in M represents a single participant's responses to each question and similarly, each column in M encodes the responses given by all the participants to a particular question.

Our goal is to use the response matrix M to uncover the correct answers g. This can be achieved if the responses of the participants to a question are correlated with the correct answer, though we do not make any specific model assumptions regarding the nature of this correlation. An aggregator is a function $f : M_{p\times q} \to [a]^q$ that takes a response matrix, produced by a set of participants for a set of questions, and outputs a proposed vector of correct answers – one for each of the considered questions. Given an input matrix $M \in M_{p\times q}$ and the vector of correct answers $g = (g_1, \ldots, g_q) \in [a]^q$ for the considered q questions, we can measure the performance of the aggregator as the proportion of questions for which the inferred answers match the correct answers: $\frac{|Q_c|}{q}$ where Q_c denotes the set of questions for which the aggregator infers the correct responses $Q_c = \{j \in Q | f(M)_j = g_j\}$.

3.1 Desiderata for Aggreators

We now describe several properties of aggregator functions that intuitively characterize minimal requirements we expect fair aggregators to fulfill.

3.1.1 Participant ordering invariance

First, we have no a priori knowledge about the relative ability of the participants. In other words, the order of the participants whose opinions we aggregate is arbitrary, thus we expect a good aggregator to show no prejudice in favour or against a participant based on their position in the response matrix. We denote by $M^{r(x\leftrightarrow y)}$ the matrix M where the rows x and y are swapped,

$$M_{i,j}^{r(x\leftrightarrow y)} = \begin{cases} M_{y,j}, & i = x \\ M_{x,j}, & i = y \\ M_{i,j}, & otherwise \end{cases} \quad (1)$$

We say an aggregator $f : M_{p\times q} \rightarrow [a]^q$ is *participant location indifferent* if for any response matrix M and x, y we have $f(M^{r(x\leftrightarrow y)}) = f(M)$.

3.1.2 Question ordering invariance

Similarly, we may not attribute special meaning to the order in which the questions were posed to the participants. In other words, the choice of a specific column in the response matrix, where a certain question is placed reflects no knowledge we have regarding the question. We denote by $M_{i,j}^{c(x\leftrightarrow y)}$ the matrix M with columns x and y swapped:

$$M_{i,j}^{c(x\leftrightarrow y)} = \begin{cases} M_{i,y}, & j = x \\ M_{i,x}, & j = y \\ M_{i,j}, & otherwise \end{cases} \quad (2)$$

Given a vector $g \in [a]^q$ (representing either the true correct answers or a suggestion made by an aggregator regarding the correct answers), we denote by $g^{x\leftrightarrow y}$ the vector with the coordinates x and y swapped. We say an aggregator $f : M_{p\times q} \rightarrow [a]^q$ is *question location indifferent* if by swapping the order of two questions x, y, we obtain the same aggregated result, except the output differs in the order of answers, for the swapped questions, i.e. the required property is: $\forall M, x, y : f(M^{c(x\leftrightarrow y)}) = (f(M))^{x\leftrightarrow y}$.

3.1.3 Answer ordering invariance

Finally, we consider assigning meaning to the identities of the possible answers. The identities of answers are the set $[a] = 1, 2, \ldots, a$. In some cases, these identities reflect no knowledge we posses regarding the dataset. For instance, in an IQ questionnaire such as the one our empirical evaluation is based on. The unique correct answer has an equal probability of being placed in any location in the answer set. In contrast, in other datasets, the order in which the answers are shown is not arbitrary. For instance, consider the case of relevance judgement queries, where users are shown the current results of a search engine for a given query and are asked which is the most relevant. The search engine ranks results from best to worst, so if it is functioning well, the best match should be placed in the first place (or at least in one of the first few places). In this case, unless answers are deliberately shuffled, the correct response is far more likely to be one of the first responses than one of the last responses.

If we know the identities of answers are chosen arbitrarily, we may want to reflect this invariance. We denote by Π_a the set of all possible permutations over the set $[a]$ (i.e. any $\pi \in \Pi_a$ is a bijection $\pi : [a] \rightarrow [a]$). Consider a given $\pi_j \in \Pi_a$, and the column vector $r_{:,j}^T = (r_{1,j}, r_{2,j}, \ldots, r_{p,j})^T$ of given responses by p participants to a certain question j. We denote by $(r_{:,j}^{\pi})^T$ the column vector consisting of the answers by the p participants where the answer identities are shuffled through π, so $\pi(r_{:,j})^T = (\pi(r_{1,j}), \pi(r_{2,j}), \ldots, \pi(r_{p,j}))^T$. Given a response matrix $M \in M_{p\times q}$, we consider the case of permuting the answer identities for each question. Given a set of permutations $h = (\pi_1, \pi_2, \ldots, \pi_q)$ (where $\pi_j \in \Pi_a$), we consider shuffling the responses with these permutations for each of the questions. We denote by M^h the matrix where the responses of the participants where shuffled through the appropriate permutation, i.e. the j'th column of M^h is $\pi_j(r_{:,j})$. Similarly, given a vector $v \in [a]^q$ and a permutation sequence $h = (\pi_1, \pi_2, \ldots, \pi_q)$, we denote the vector where each element was shuffled by the respective permutation as $h(v) = (\pi_1(v_1), \pi_2(v_2), \ldots, \pi_q(v_q))$. We say an aggregator is *answer identity indifferent* if for any response matrix M and $h = (\pi_1, \pi_2, \ldots, \pi_p)$ we have $(f \circ h)(M) = (h \circ f)(M)$.

4 DeepAgg - TRAINING A NEURAL NETWORK AS AN AGGREGATOR

Our goal is to train a neural network as an aggregator $f_n : M_{p\times q} \rightarrow [a]^q$, following the desiderata of Section 3.1. We thus aim to generate an aggragtor that is participant location indifferent, question location indifferent and answer identity indifferent. This is a supervised learning problem, where the input is a response matrix $M \in M_{p\times q}$ and the desired output is the set of correct answers.

4.1 Constructing a Synthetic Training Set

The desired aggregator function is a complex mapping, so training a neural network requires a large training set. Given an infinite supply of participants and questions (along with their correct answers), we could repeatedly source q questions and p participants from the infinite sup-

ply source, pose the questions to the participants, collect the answers and thus obtain a large training set. However, in real-world settings we have a very limited supply of participants and questions.

Our solution relies on *synthetic data generation* through subsampling. We first take an original dataset, consisting of the responses of $p' > p$ participants to a set of $q' > q$ questions. We then select q questions at random from the q' available questions, and p participants of the p' available participants, and use only their responses. A single choice of q questions, $\mathcal{I}_q \in C(q', q)$ [3], and p participants, $\mathcal{I}_p \in C(p', p)$, results in a subsampled input matrix $M_{\mathcal{I}_p,\mathcal{I}_q} \in M_{p\times q}$. We repeat this process many times to generate a training dataset $D = \{(M_{\mathcal{I}_p,\mathcal{I}_q}, g_{\mathcal{I}_q})\}_{\mathcal{I}_q\in C(q',q),\mathcal{I}_p\in C(p',p)}$ where $g_{\mathcal{I}_q}$ denoted the correct responses for the subset of questions $\mathcal{I}_q$. In total we will have $\binom{q'}{q} \cdot \binom{p'}{p}$ choices of training instances. This augmented dataset enables us to transform a moderately small original dataset into one containing enough sub-sampled training examples to successfully train a neural net.

4.2 Adhering to Desiderata

Training a neural network involves optimizing the network's weight parameters based on the training set. The network could thus overfit the parameters to various properties of the training set that fail to generalize to the test set. For instance, if one answer is more common then others in the training set, a trained network might reflect this and select weights leading to this answer being chosen more often than others. When such trends are specific to the training data and do not occur on the test set, this overfitting is likely to lead to reduced performance. The Desiderata in Section 3.1 reflect assumptions regarding how the data was generated and the expected behavior of a good aggregator. Adhering to these properties avoids certain forms of overfitting. How should we design and train a neural network so it would achieve these properties and avoid overfitting?

One possibility is relying on the synthetic data generation process, and applying various *synthetic dataset perturbations*. Note that out of the three invariance properties identified in Section 3.1, the constructed dataset D encodes the first two. Thus in order to enforced the third invariance, we can permute the answer identities for each question in every subsampled data, by taking a set of random answer identity permutations $h = (\pi_1, \pi_2, \ldots, \pi_q)$ (where $\pi_j \in \Pi_a$), and shuffling the responses with these permutations for each of the questions. During the synthetic dataset construction, we generate a subsampled matrix $(M_{\mathcal{I}_p,\mathcal{I}_q}, g_{\mathcal{I}_q}) \in D$. Now we can expand this dataset by shuffling the answer identities via a random permutation h: $(h\circ(M_{\mathcal{I}_p,\mathcal{I}_q}, g_{\mathcal{I}_q})) = (h\circ(M_{\mathcal{I}_p,\mathcal{I}_q}), h\circ(g_{\mathcal{I}_q}))$. [4] As the answers identities are randomly shuffled, even if one answer identity is more frequently the correct answer in the original dataset, in the synthetic dataset any answer has an equal probability of being the correct answer. Hence, training on the synthetic dataset should result in an answer identity indifferent aggregator. Thus we get a large training dataset $\mathcal{D} = \{(h \circ (M_{\mathcal{I}_p,\mathcal{I}_q}, g_{\mathcal{I}_q}))|h \sim \Pi_a^q, (M_{\mathcal{I}_p,\mathcal{I}_q}, g_{\mathcal{I}_q}) \in D\}$.

An alternative approach to synthetic dataset perturbations is relying on the *chosen features* to achieve the desiderata: we simply create features that *abstract away* the irrelevant information. Under this approach we first construct the synthetic dataset D. We then take a training instance $(M_{\mathcal{I}_p,\mathcal{I}_q}, g_{\mathcal{I}_q}) \in D$, construct features representing this instance, denoted as $\phi(M_{\mathcal{I}_p,\mathcal{I}_q})$. Rather than training a neural network on the "raw" input $M \in D$, we train it on the feature representation, $\phi(M)$. Although any neural network must receive some *representation* of the training instance in some form, we will construct the feature representation in a way that eliminates information about participant locations, question locations or answer identities. More formally, we say a feature representation ϕ is participant location indifferent if changing the location of a participant in the matrix has no influence on the generated representation, so $\phi(M^{r(x\leftrightarrow y)}) = \phi(M)$ (for any $M \in D; x, y \in \mathcal{I}_p$). Similarly, we say a feature representation is question location indifferent if $\phi(M^{c(x\leftrightarrow y)}) = \phi(M), \forall M \in D; x, y \in \mathcal{I}_q$, and say that it is answer identity indifferent if $\phi(h \circ M) = \phi(M), \forall M \in D$,$h = (\pi_1, \pi_2, \ldots, \pi_p)$. [5]

4.3 An Overview of Our Approach

Our approach is indeed based on choosing a feature representation that adheres to our desiderata. Earlier work shows that while stronger aggregators do exist, majority vote is already powerful in aggregating crowdsourced responses [2]. Our architecture thus begins by using the majority vote rule to determine an initial answer sheet, and constructing features that describe participants and questions based on this initial chosen answer set; we then *gradually refine* this answer sheet, using two building blocks.

The first block is a neural network that predicts the probability that a participant would respond correctly to a specific question. We refer to these probabilities as "success probabilities". The input to the block is a proposed answer sheet, reflecting the "best guess" of the correct answer for each question. Given this answer sheet, the block constructs our feature representation of the participants and

[3] We denote by $C(n, k)$ the set of k-combinations that can be form using n elements

[4] See the definitions of these operators in Section 3.1.

[5] These properties of a *feature representation* are similar to those of an aggregator function, except in the case of a feature representation we refer to an *encoding* of the important bits of the input matrix that allow determining how to best aggregate the data, rather than to the end result consisting of the chosen answer for each question.

questions, and applies the neural network to compute the success probabilities.

The second building block focuses on a single question, and receives as input these success probabilities of the participants as well as the answer chosen by each participant, and outputs the chosen answer for the question. We propose two alternatives for constructing the second block, which both work by computing the total support for each possible answer, then outputting the answer with the strongest support. One alternative is using a neural network, and the other is a deterministic formula that weighs the success probabilities using a simplistic probabilistic model of the behavior of participants. [6]

Combining the two building blocks, we obtain a method for refining an answer sheet. The first block takes the current answer sheet and the responses of the participants, and computes the success probabilities; the second block uses these success probabilities to output an improved answer sheet. We call the application of the two building blocks a *refinement step*. We initialize the system with an answer sheet computed by majority vote, and perform multiple refinement steps to generate our final answer sheet. [7]

We now describe the features used to represent the participants and questions, then describe the building blocks and the overall network architecture in more detail.

4.4 Feature Representation of the Input

Consider an input matrix $M \in M_{p\times q}$ consisting of the responses of p participants to a set of q questions, and a proposed set of answers to these questions $g' \in [a]^n$. An element g'_i can be thought of as the current best guess for the answer to question i (our algorithms initialize g'_i as the majority vote to question i in the subsampled matrix). We refer to the vector g' of proposed responses as an answer sheet. A rough estimate for the success or ability of a participant i is the proportion of questions to which they responded correctly, assuming that the answer sheet g' is indeed correct: $a_i = \frac{|\{j\in[q]|M_{i,j}=g'_j\}|}{q}$. Similarly, a rough estimate for a difficulty of a question is the proportion of students who failed to provide the correct response to it: $d_j = 1 - \frac{|\{i\in[s]|M_{i,j}=g'_j\}|}{s}$. Clearly, the quality of the features a_i (for a participant i) and d_j (for a question j) depend on the answer sheet g': the higher the quality of g', the less noisy these features are.

The above features allow predicting whether a participant is likely to correctly answer a question, and can also be used to rank participants based on their ability or questions by their difficulty. Consider sorting the questions by their estimated difficulty, d_j, and partitioning this set into k parts, $D_1, \ldots, D_k$ by the estimated difficulty. For example, for $k = 2$ we partition the questions into two parts by estimated difficulty:the easiest half of the questions are D_1 (with the lower difficulty estimate d_j), and the hardest half of the questions are D_2 (with the highest difficulty estimate d_j). We can estimate the ability of a participant using only the questions in one part of the partition. For instance, for $k = 2$ the performance of participant i on the easiest questions D_1 is $a_i^1 = \frac{|\{j\in D_1|M_{i,j}=g'_j\}|}{|D_1|}$, and their performance on the hardest questions is $a_i^2 = \frac{|\{j\in D_2|M_{i,j}=g'_j\}|}{|D_2|}$. More generally, we choose a number of parts (the "width" k), and denote $a_i^t = \frac{|\{j\in D_t|M_{i,j}=g'_j\}|}{|D_t|}$ (where $t \in \{1, 2, \ldots, k\}$, and $|D_t| = \frac{q}{k}$, assuming that k divides q).

Consider partitioning the questions into two parts, the easy questions D_1 and hard questions D_2 (with difficulty estimated using the current answer sheet g'). We'd expect a student i to do better on the easy questions than on the hard questions, so a_i^1 is higher and a_i^2 is lower. If we have a student where a_i^1 is lower than a_i^2, we might infer that their success on the harder questions is due to luck rather than skill (and predict him to be less successful than a student v of identical average skill but where a_v^1 is higher than a_v^2). Thus $a_i^1, \ldots, a_i^k$ can serve as additional features providing further useful information, beyond just the average skill.

Similarly to partitioning the questions by their difficulty to build a more fine-grained representation of students, we can partition the students by ability to generate a better representation for questions. We sort the students by their estimated ability, a_i, and partition this set into k' parts, $E_1, \ldots, E'_k$. We estimate the difficulty of a question using only the students in one part of the partition (e.g. the difficulty of a question for the strong and weak students). We denote $d_j^t = 1 - \frac{|\{i\in E_t|M_{i,j}=g'_j\}|}{|E_t|}$ (where $t \in \{1, 2, \ldots, k'\}$, and $|E_t| = \frac{s}{k'}$, assuming that k' divides s).

The goal of our first building block is to take an input subsampled matrix M and a proposed answer sheet g' and determine the probability of a student i to correctly answer question j. We represent the students and questions using the above features: $\phi_{i,j}(M) = (a_i, a_i^1, \ldots, a_i^k, d_j, d_j^1, \ldots, d_j^{k'})$ (where the features are computed using g' as the answer sheet). We note that the representation ϕ is answer identity indifferent, participant location indifferent and question location indifferent.

4.5 Predicting Whether a Participant Will Answer a Question Correctly

We use a neural network which takes a representation of a single user and a single questions, and predicts the prob-

[6] In the empirical analysis in Section 5 we show both perform well on our dataset (with no significant performance difference between the two).

[7] If a refinement step does not change the answer sheet, we have reached a fixed point of the refinement function, and can terminate the process early.

ability that the user would correctly answer the question. The input to the network is $\phi_{i,j}(M)$ (where $i \in [p]$ and $j \in [q]$), as defined in Section 4.4. We have used the dimensions $k = k' = 3$, so both a user and a question are represented as 4 real numbers, meaning each input is a vector $\phi_{i,j}(M) \in \mathbb{R}^8$. Our network consists of three fully connected layers, each performing a linear transformation then a $\tanh$ operation. The first linear transformation maps the input to a hidden layer with a hidden state of size $h = 10$ neurons, and the following layers maintain this hidden state size. Following these layers, our network linearly maps the final layer into a single neuron.

Given a single subsampled matrix M and given the correct answer sheet g (consisting of the known correct answers to the subsampled questions), we use $y_{i,j}$ as an indicator variable denoting whether the participant i answered question j correctly, so $y_{i,j} = 1$ if $M_{i,j} = g_j$ and $y_{i,j} = 0$ otherwise. [8] Each subsampled matrix results in $p \cdot q$ representations for participant-question pairs, $\phi_{i,j}(M)$, and in $p \cdot q$ indicator variables $y_{i,j}$. Applying the network on all the representations of participant-question pairs, we obtain $p \cdot q$ activations of the final layer, denoted as $\hat{y}_{i,j}$. We use the cross entropy loss:

$$\mathcal{L} = \sum_{i=1}^{p}\sum_{j=1}^{q}[y_{i,j}\log\hat{y}_{i,j} + (1-\hat{y}_{i,j})\log(1-\hat{y}_{i,j})]$$

We train the network by randomly subsampling $n = 10,000$ matrices along with the correct responses to the questions. Once the network is trained, it can receive a representation $\phi_{i,j}$ of a participant i and a question j, and its output $\hat{y}_{i,j}$ can be interpreted as the probability that the participant would answer the question correctly. Figure 1 illustrates the structure of this neural network.

4.6 Computing Support for a Given Answer

The second building block in our approach considers a single question and the responses of p participants to that question. The goal of the block is to provide a score for each answer, such that the correct answer would receive the highest score. We refer to this as computing the aggregate support in favor of an answer. The output of the block is thus a vector of scores $s = (s_1, s_2, \ldots, s_a)$ where s_t is the score of answer t. We propose two alternatives for the construction of this building block.

The first alternative we propose is a neural network. Given the probability of a participant i to correctly answer the question, denoted b_i for brevity [9] and the responses of each participant to the question, denoted $m_i = r_{i,j}$ for the question j, we construct a $a \times p$ matrix U representing a probability distribution over the a possible answers.

$$U_{i,k} = \begin{cases} b_i, & k = m_i \text{ (participant's chosen answer)} \\ \frac{1-b_i}{a-1}, & otherwise \end{cases} \tag{3}$$

This is an alternative to a "one-hot" encoding of $\{b_i\}_{i=1:p}$, reflecting the probability of a participant to choose each answer, assuming that the correct answer is indeed the one they chose, and that each incorrect answer is equally likely (the remaining mass $1 - b_i$ spread evenly across them).

The input to the network are the rows in matrix $U = (u_1^T, \cdots, u_a^T)$ – one for each possible answer. As the identity of each answer does not matter, we train a function f that takes a answer probability vector u and produces a score, s, for this answer vector. To learn the function we use a network of 3 fully connected layers, each consisting of a linear transformation followed by a $\tanh$ non-linearity, and a hidden size of $h = 15$, followed by another linear layer mapping the h neurons to a score s. We use the same mapping f to produce scores for all a possible answers, leading to $(s_1, \ldots, s_a) = (f(u_1), \ldots, f(u_a))$.

Each subsampled matrix used to train this network consists of q questions, where each question has an encoding, U, of size $p \cdot a$ representing the responses of the participants. Further, each subsampled matrix is considered along the vector of correct responses to each question $(g_1, \ldots, g_q)$. As we wish to optimize the network parameters so that the correct answer g_j to question j would receive the highest score s_j, the final layer of our network is a softmax operator: $\frac{\exp(s_i)}{\sum_{i=1}^{a}\exp(s_i)}$, and we use the cross-entropy loss. Figure 2 illustrates the structure of this neural network.

The second alternative we propose is a deterministic scor-

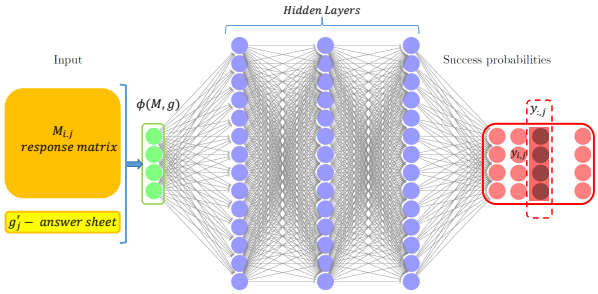

Figure 1: Visualization of the first building block: predicting success probabilities $y_{i,j}$, given a response matrix $M \in D$ and current answer sheet $g' = (g'_1, \cdots, g'_q)$

[8] Note that while training the network we use the ground truth correct answers. However, once the neural network is trained, it does not require this information, and can be used for any dataset. In our empirical analysis we train the network using one dataset, and use it on a completely separate dataset (both participants and questions that were not shown during training).

[9] Replacing the notation $\hat{y}_{i,j}$ as the question index is redundant

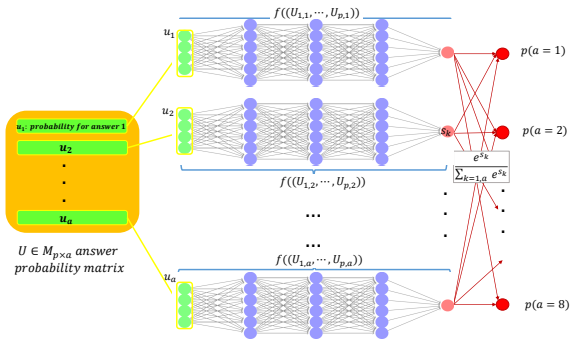

Figure 2: Visualization of the second building block: predicting the answer for a particular question j. Given the participants responses $M_{:,j}$ and the inferred probabilities of participants to be right or wrong $y_{:,j}$, we construct an answer probability matrix U as described in Eq. 3. Then for each possible answer k, we compute a score based on the participants' answer probabilities u_k: $s_k = f(u_k)$. These scores go through a `softmax` operator and finally we choose the answer with the highest probability.

ing function, based on a simple probabilistic model for the way in which the responses are chosen by the participant. Suppose that a priori each answer is equally likely to be the correct answer (reflecting answer identity invariance). Thus the correct answer is a random variable A, with a uniform distribution over the support $\{1,\ldots,a\}$. Assume that the probability of a participant i to choose the right answer to the question is b_i (given as the block's input), and that if i fails to choose the correct answer, they choose an answer uniformly at random from the remaining $a-1$ answers. Finally, we assume that the participants are independent of one another. The responses of the participants are thus an observed random variable R over the support $[a]^p$.

Consider the case where the correct answer to the question as $r \in [a]$. Given the observed responses R, we can partition the participants into two sets: the participants who responded with the answer r as X_r (the correct participants), and those who responded with some other answer X_o (the incorrect participants). To obtain the observed responses R, each of the correct participants must have chosen the correct answer (with probability b_i), and each of the incorrect participant must have failed to chose the correct answer (with probability $1-b_i$), and then choose exactly the incorrect answer they chose (there are $a-1$ incorrect responses and each is equally probable). Thus, under our model, the probability of obtaining the observed responses R is:

$$P(R|A=r) = \prod_{f\in X_r} b_f \cdot \prod_{g\in X_o} \frac{1-b_g}{a-1} \qquad (4)$$

The goal of the second building block is choosing the best response for a question given the input parameters $b1,\ldots,b_p$ and the observed responses R. By Bayes' theorem we have $P(A|R) = \frac{P(R|A)\cdot P(A)}{P(R)}$. Given the observed responses R and our assumed model, we seek the most probable answer $\arg\max_a P(A=a|R) = \frac{P(R|A=a)\cdot P(A=a)}{P(R)}$. As $P(R)$ is a normalizing constant and as we assumed that for any a we have $P(A=a)=\frac{1}{a}$, we have $\arg\max_a P(A=a|R) = \arg\max_a P(R|A=a)$. We can thus simply apply Equation 4 to compute the score $P(R|A=r)$ for each possible response $r \in [a]$, and return the answer with the maximal score.

4.7 Iterative Refinement

Section 4.4 discusses how we take a subsampled matrix M and a proposed answer sheet g'_0, and output a feature representation $\phi(M,g')$ for each user and question. Section 4.5 discusses how we take the feature representation and apply a neural network to predict $y_{i,j}$, the probability of each participant i to answer any question j. We call this step $cor(\phi(M,g'))$. Finally, section 4.6 discusses how we take the success probabilities $y = (y_{1,1}, y_{1,2}, \ldots, y_{1,q}, y_{2,1}, \ldots, y_{p,q})$ and the subsampled response matrix M and generate and refined answer sheet g' (either using a trained neural network, or through the formula on equation 4). We call this step $ref(y,M)$.

Our method, called DeepAgg, simply involves applying multiple such iterations (the number of iterations is the parameter $nIter$). Algorithm 1 describes this process.

Data: Reponse matrix $M \in M_{p\times q}$
Result: Aggregated responses $g' \in [a]^q$
$g' \leftarrow \text{Maj}(M)$ // Majority vote initialization ;
for $i \leftarrow 1$, *nIter* **do**
  $y \leftarrow cor(\phi(M,g'))$;
  $g' \leftarrow ref(y,M)$;
end
return g' ;

Algorithm 1: DeepAgg: Iterative Refinement

5 EMPIRICAL ANALYSIS

We empirically examine the DeepAgg method of Section 4, and evaluate its performance against both the majority vote aggregator and the DARE Bayesian aggregator of Bachrach et al. [2], using the same dataset described in that paper. This dataset consists of the responses of participants to the Raven's Standard Progressive Matrices (SPM) IQ test [15], an intelligence screening test. This test is a multiple choice questionnaire, consisting of $q' = 46$ questions, each with eight possible answers. [10] SPM is a popular intelligence

[10] The full test has 4 parts in increasing difficulty. We removed the easiest questions to focus on the more interesting cases where there is not a consensus between participants. Note that including all questions does not alter the qualitative conclusions of this paper.

test, that has been used for research purposes, clinical assessment and even military personnel screening [16]. The dataset contains the responses of $p' = 746$ participants, who took the test in the 2006, in the process of establishing the ability norms for the British market [15]. [11]

To conduct our experiments, we partitioned the dataset's questions into two parts, Q_a and Q_b, each with half (23) of the dataset's questions (where $Q_a \cup Q_b = \emptyset$). Similarly, we partitioned the participants into two parts, P_a and P_b, each with half (373) of the dataset's participants (with $Q_a \cup Q_b = \emptyset$). We trained DeepAgg using subsampled data from the responses of the participants in P_a to the questions in Q_a, and evaluated the performance using subsampled data from the responses of the participants in P_b to the questions in Q_b. This guarantees that the training and testing are done on separate datasets: not only are the training instances different from the test instances, effectively we are testing the aggregator on a completely unseen dataset.

5.1 Aggregation Quality

DeepAgg aggregates the responses of multiple participants and outputs the predicted correct answer for each question, similarly to majority vote or the DARE aggregator of Bachrach et al. [2]. We now compare the performance of these approaches. Our quality metric for an aggregator is simple. Given a response matrix $M^i \in M_{p \times q}$, an aggregator returns a vector $g'^i \in [a]^q$. Given the the set of actual correct responses $g^i \in [a]^q$, we denote the number of questions where the aggregator was correct as $c(g') = |\{j | g'^i_j = g^i_j\}|$. We denote the proportion of questions that were aggregated successfully as $s^i(g') = \frac{c^i(g'^i)}{q}$.

Figure 3 shows the quality of majority vote, DARE and our DeepAgg, as a function of the available date. When an aggregator has more available responses for each question, we expect it to achieve a better performance as it has more available data. The x-axis of Figure 3 shows the number of responses per question (the number of participants in the subsampled response matrix), and the y-axis shows the aggregation quality $\frac{1}{k}\sum_{i=1}^{k} s^i(g'^i)$, across $k = 10,000$ runs (each of which is a random subsampled response matrix).

Figure 3 shows that all methods achieve a better result as more data is available, but that the returns diminish as the number of participants increases (in agreement with the results reported in the work of Bachrach et al. [2]). For all data regimes, DeepAgg outperforms DARE.

The plot in Figure 3 was created with DeepAgg using equation 4 as the second building block. We have also run DeepAgg with the neural net implementation for the second block, with an almost identical performance, indicating that both are reasonable choices for that block.

[11] We thank the Psychometrics Centre of the University of Cambridge for making this dataset available for this research.

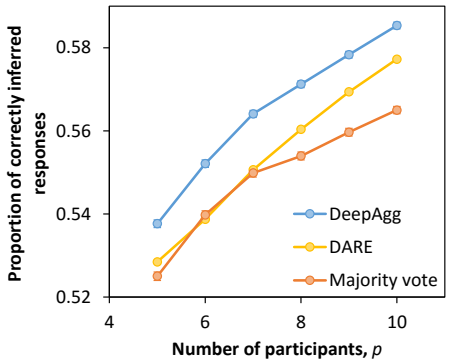

Figure 3: The performance of DeepAgg and alternative aggregators as a function of the size of the data

DeepAgg performs multiple iterations of improving the answer sheet, as described in Algorithm 1. Figure 4 shows the effect of the number of iterations on the quality of aggregation. The x-axis is the number of iterations ($nIter$ in Algorithm 1) and the y-axis is the average aggregation quality $\frac{1}{k}\sum_{i=1}^{k} s^i(g'^i)$ across $k = 10,000$ runs (with random subsampled response matrices). We find that performing multiple iterations improves the quality of aggregation, but that this improvement diminishes as the number of iterations increases. One possible cause for these diminishing returns is that if the answer sheet g' does not change following an iterations, we have reached a fixed point of this function, so there is no point in performing additional iterations.

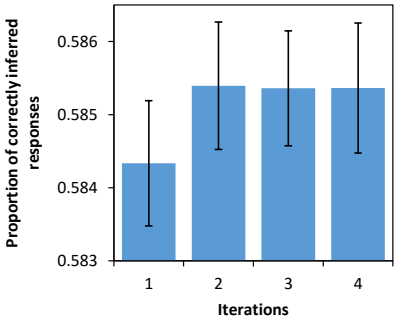

Figure 4: The effect of the number of DeepAgg iterations on aggregation quality

5.2 Evaluating the ability of participants

While the main stated goal of the DeepAgg aggregator is inferring the set of correct answers g, a common use-case in crowdsourcing settings is deciding which of the participants are good at the task and which are not.

One possible measure of the ability of a participant i given a response matrix M and a set of correct answers $g \in [a]^q$, is the proportion of questions which i answered correctly: $a_i = \frac{|\{j \in [q] | M_{i,j} = g_j\}|}{q}$. If we are not given the ground truth

set of responses, g, we could use an approximate answer sheet g', and use $a'_i = \frac{|\{j \in [q] | M_{i,j} = g'_j\}|}{q}$ as an approximation. Clearly, the quality of this estimator depends on how well g' approximates g. As DeepAgg allows inferring an answer sheet g' given the response matrix M, one could thus use it as a tool for inferring the ability of participants.

Figure 5 investigates how well DeepAgg performs in inferring the ability of participants. This is a density plot which was generated by examining many data points, each describing a single random participant in a run of DeepAgg on a random subsampled matrix. The x-axis of each point represents the true ability of the participant, a_i, and the y-axis is the estimated ability of the participant, a'_i (using an answer sheet g' inferred using DeepAgg). The figure shows the density of the sampled points in each area on the chart.

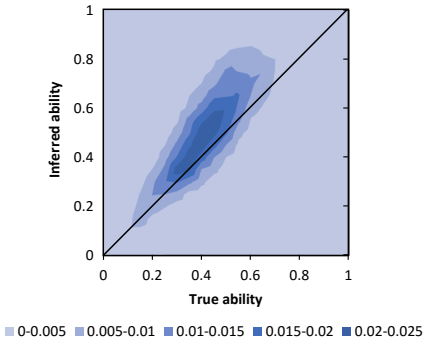

Figure 5: Using DeepAgg to infer the ability of participants

A perfect aggregator would result in a figure where points rest on the $y = x$ axis, and a Pearson correlation of 1. Figure 5 shows a very strong correlation between the inferred and true ability of each participant, even with few iterations. The Pearson correlation between the true ability of a participant and the inferred ability of that participant is $r = 0.79$, which is high considering the limited amount of data available to DeepAgg. This indicates that DeepAgg is a powerful tool not only in aggregating responses, but also for identifying skilled and less-skilled participants.

5.3 The Need for Computing Features

A major advantage of deep learning is its ability to learn feature representations without resorting to hand-crafted features [3, 19]. One might ask why we have used a method that relies on the features discussed in Section 4.4. A simpler architecture could use a deep neural net which simply operates on the input subsampled matrix M (for instance using a "one-hot" encoding for the chosen response of each participant on each question). We have indeed tested such an architecture, showing the performance it achieves to only match majority vote. A very deep network is theoretically capable of performing the overall computation we have done using DeepAgg. However, as this requires a very deep network, training the network using stochastic gradient descent (or similar variants) must search through an extremely large and complicated parameter space. The fact that a direct architecture does not achieve the high performance of DeepAgg indicates that the current optimization methods are incapable of finding this solution.

The computation in DeepAgg is quite deep: we apply a 3 layer deep network twice for each iteration (each building block is a network), so with even as few as 3 iterations this is a 18 layer deep architecture. Further, our computation relies on computing the desired features once in each iterations. Our ability to successfully train the network stems from the *constant supervision* we apply: a loss can be computed after every building block using the ground truth. In other words, once every basic building block, we apply a loss using the ground truth answer sheet g, allowing us to find excellent optimized parameters for each block.

6 LIMITATIONS AND CONCLUSIONS

We presented DeepAgg, an approach for aggregating crowdsourced responses, based on a deep neural network. Our empirical analysis shows that DeepAgg has a superior performance over the majority aggregator and a more sophisticated Bayesian approach. Our approach has some **inherent limitations**. Training the network requires taking an initial dataset and repeatedly sub-sampling parts of it to generate synthetic training examples. This training procedure is a computationally expensive calculation, which yields an aggregator taking the responses of p participants to q questions. Once the aggregator is trained, applying it to a new training instance is has a relatively low runtime complexity. However, if the input dimensions p or q are changed, the training process needs to be repeated to create a new aggregator. Further, we used a simple network architecture. A more elaborate structure could potentially improve performance. Finally, our method is desigend for the **complete data case**. It is difficult to adapting our method to the case of incomplete data, where some of the participants have only answered some of the questions. [12]

Several issues remain open for future research. How can our procedure be modified to handle the case of missing responses, where some participants may only provide responses to a subset of the questions? While it is easy to encode this in the input to the network, this may have a large impact of the quality of the aggregator. Second, is it possible to extend our approach to an active learning scenario, where we have control over the next question to ask a participant? Finally, could we take an aggregator training for certain input dimensions and convert it into an aggregator for other input sizes, without retraining a network?

[12] In contrast, Bayesian approaches are usually robust to missing data (for instance, in approaches based on graphical models, these can simply be treated as unobserved random variables).

References

[1] Y. Bachrach, T. Graepel, G. Kasneci, M. Kosinski, and J. Van Gael. Crowd IQ - aggregating opinions to boost performance. In *AAMAS*, 2012.

[2] Yoram Bachrach, Thore Graepel, Tom Minka, and John Guiver. How to grade a test without knowing the answers—a bayesian graphical model for adaptive crowdsourcing and aptitude testing. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1183–1190, 2012.

[3] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[4] Nilesh Dalvi, Anirban Dasgupta, Ravi Kumar, and Vibhor Rastogi. Aggregating crowdsourced binary ratings. In *Proceedings of the 22nd international conference on World Wide Web*, pages 285–294. International World Wide Web Conferences Steering Committee, 2013.

[5] M.J.A.N. de Caritat et al. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. L'imprimerie royale, 1785.

[6] Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: a survey. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(5):845–869, 2014.

[7] L. Getoor, N. Friedman, D. Koller, A. Pfeffer, and B. Taskar. 5 probabilistic relational models. *Statistical relational learning*, page 129, 2007.

[8] Allan Gibbard. Manipulation of voting schemes: a general result. *Econometrica: journal of the Econometric Society*, pages 587–601, 1973.

[9] G. Kasneci, J. Van Gael, R. Herbrich, and T. Graepel. Bayesian knowledge corroboration with logical rules and user feedback. In *ECML/PKDD*, 2010.

[10] M. Kosinski, Y. Bachrach, G. Kasneci, J. Van-Gael, and T. Graepel. Crowd IQ: Measuring the intelligence of crowdsourcing platforms. In *ACM Web Sciences*, 2012.

[11] A. McLennan. Consequences of the condorcet jury theorem for beneficial information aggregation by rational agents. *American Political Science Review*, pages 413–418, 1998.

[12] D.M. Pennock and R. Sami. Computational aspects of prediction markets, 2007.

[13] Chris Piech, Jonathan Huang, Zhenghao Chen, Chuong Do, Andrew Ng, and Daphne Koller. Tuned models of peer assessment in moocs. *arXiv preprint arXiv:1307.2579*, 2013.

[14] Guo-Jun Qi, Charu C Aggarwal, Jiawei Han, and Thomas Huang. Mining collective intelligence in diverse groups. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1041–1052. International World Wide Web Conferences Steering Committee, 2013.

[15] J.C. Raven. Standard progressive matrices plus.

[16] J.C. Raven. The raven's progressive matrices: Change and stability over culture and time. *Cognitive Psychology*, 41(1):1–48, 2000.

[17] V.C. Raykar, S. Yu, L.H. Zhao, G.H. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *JMLR*, 2010.

[18] Mahyar Salek, Yoram Bachrach, and Peter Key. Hotspotting-a probabilistic graphical model for image object localization through crowdsourcing. In *AAAI*, 2013.

[19] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[20] A. Sen. Social choice theory. *Handbook of mathematical economics*, 3:1073–1181, 1986.

[21] Bar Shalem, Yoram Bachrach, John Guiver, and Christopher M Bishop. Students, teachers, exams and moocs: Predicting and optimizing attainment in web-based education using a probabilistic graphical model. In *Machine Learning and Knowledge Discovery in Databases*, pages 82–97. Springer, 2014.

[22] James Surowiecki. *The wisdom of crowds*. Anchor, 2005.

[23] P. Welinder, S. Branson, S. Belongie, and P. Perona. The multidimensional wisdom of crowds. In *NIPS*, 2010.

[24] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, and J. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. *NIPS*, 22:2035–2043, 2009.