Learning Algorithms for Active Learning

Philip Bachman [* 1]   **Alessandro Sordoni** [* 1]   **Adam Trischler** [1]

Abstract

We introduce a model that learns active learning algorithms via metalearning. For a distribution of related tasks, our model jointly learns: a data representation, an item selection heuristic, and a prediction function. Our model uses the item selection heuristic to construct a labeled support set for training the prediction function. Using the Omniglot and MovieLens datasets, we test our model in synthetic and practical settings.

1. Introduction

For many real-world tasks, labeled data is scarce while unlabeled data is abundant. It is often possible, at some cost, to obtain labels for the unlabeled data. In active learning, a model selects which instances to label so as to maximize some combination of task performance and data efficiency. Active learning is motivated by the observation that a model may perform better while training on less labeled data if it can choose the data on which it trains (Cohn et al., 1996). E.g., in SVMs (Schölkopf & Smola, 2002) only the support vectors affect the decision boundary. If one could identify the support vectors in advance, the classifier trained on the resulting set of examples would obtain the same decision boundary with less data and computation.

Active learning can benefit a variety of practical scenarios. For example, preference information for a new user in a movie recommender system may be scarce, and recommendations for the new user could be improved by carefully selecting several movies for her to rate (Sun et al., 2013b; Houlsby et al., 2014; Aggarwal, 2016). Likewise, collecting labels for a medical imaging task may be costly because it requires a specialist (Hoi et al., 2006), and the cost could be reduced by carefully selecting which images to label.

Various heuristics for selecting instances to label have been proposed in the active learning literature, such as choos-

ing the instance whose label the model is most uncertain about, or the instance whose label is expected to maximally reduce the model's uncertainty about labels for other instances (Gilad-Bachrach et al., 2005; Settles, 2010; Houlsby et al., 2011). We propose moving away from engineered selection heuristics towards *learning* active learning algorithms end-to-end via *metalearning*. Our model interacts with labeled items for many related tasks in order to learn an active learning strategy for the task at hand. In recommendation systems, for example, ratings data for existing users can inform a strategy that efficiently elicits preferences for new users who lack prior rating data, thus *bootstrapping* the system quickly out of the cold-start setting (Golbandi et al., 2010; 2011; Sun et al., 2013a; Kawale et al., 2015). A learned active learning strategy could outperform task-agnostic heuristics by sharing experience across related tasks. In particular, the model's (i) data representation, (ii) strategy for selecting items to label, and (iii) prediction function could all co-adapt. Moving from pipelines of independently-engineered components to end-to-end learning has lead to rapid improvements in, e.g., computer vision, speech recognition, and machine translation (Krizhevsky et al., 2012; Hannun et al., 2014; He et al., 2016; Wu et al., 2016).

We base our model on the Matching Networks (MN) introduced by Vinyals et al. (2016). We extend the MN's one-shot learning ability to settings where labels are not available *a priori*. We cast active learning as a sequential decision problem: at each step the model requests the label for a particular item in a pool of unlabeled items, then adds this item to a labeled support set, which is used for MN-style prediction. We train our model end-to-end with backprop and reinforcement learning. We expedite the training process by allowing our model to observe and mimic a strong selection policy with oracle knowledge of the labels.

We demonstrate empirically that our proposed model learns effective active learning algorithms in an end-to-end fashion. We evaluate the model on "active" variants of existing one-shot learning tasks for *Omniglot* (Lake et al., 2015; Vinyals et al., 2016; Santoro et al., 2016), and show that it can learn efficient label querying strategies. We also test the model's ability to learn an algorithm for bootstrapping a recommender system using the *MovieLens* dataset, showing it holds promise for application in more practical settings.

---

[*]Equal contribution [1]Microsoft Maluuba, Montreal, Canada. Correspondence to: P. Bachman <phbachma@microsoft.com>, A. Sordoni <alsordon@microsoft.com>.

2. Related Work

Various heuristics have been proposed to guide the selection of which examples to label during active learning (Settles, 2010). For instance, Lewis & Gale (1994) and Tong & Chang (2001) developed policies based on the confidence of the classifier, while Gilad-Bachrach et al. (2005) used the disagreement of a *committee* of classifiers. Houlsby et al. (2011) presented an approach based on Bayesian information theory, in which examples are selected in order to maximally reduce the entropy of the posterior distribution over classifier parameters.

The idea of learning an active learning algorithm end-to-end, via *meta* active learning, was recently investigated by Woodward & Finn (2016). Building on the memory-augmented neural network (MANN) (Santoro et al., 2016), the authors developed a *stream-based* active learner. In stream-based active learning the model decides, while observing items presented in an exogenously-determined order, whether to predict each item's label or to pay a cost to observe its label. Our proposed model instead falls into the class of *pool-based* active learners, i.e. it has access to a static collection of unlabeled data and selects both the items for which to observe labels, and the order in which to observe them.

Active learning can be useful when the cost incurred for labeling an item may be traded for lower prediction error, and where the model must be data efficient (e.g. in medical imaging (Hoi et al., 2006)). We explicitly train our model to balance between task performance and labeling cost. In this sense, we build an *anytime* active learner (Zilberstein, 1996), with the model trained at each step to output the best possible prediction on the evaluation set.

Our model builds on the matching-networks (MN) architecture presented by Vinyals et al. (2016), which enables "one-shot" learning, i.e. learning the appearance of a class from just a single example of that class (Santoro et al., 2016; Koch, 2015). Vinyals et al. (2016) assume that at least one example per class exists in the labeled support set available to the model. Confronted with the harder task of composing a labeled support set from a larger pool of unlabeled examples, we show that the active learning policy learnt by our model obtains, in some cases, an equally effective support set. As in the recent one-shot learning work of Santoro et al. (2016) and Vinyals et al. (2016), and the active learning work of Woodward & Finn (2016), we evaluate our model on the *Omniglot* dataset. This dataset was developed for the foundational one-shot learning work of Lake et al. (2015), which focused on probabilistic program induction.

The cold-start problem is ubiquitous in recommendation systems (Aggarwal, 2016; Lika et al., 2014; Harpale & Yang, 2008; Sun et al., 2013b; Elahi et al., 2016). Instead of bootstrapping from a cold-start by randomly selecting items for a user to rate, an active learner asks for particular items to help learn a strong user model more quickly. In model-free strategies (Rashid et al., 2008), items are selected according to general empirical statistics such as popularity or informativeness. These approaches are computationally cheap, but lack the benefits of adaptation and personalization. Proposals for learning an adaptive selection strategy have been made in the form of Bayesian methods that learn the parameters of a user model (Houlsby et al., 2014; Harpale & Yang, 2008), and in the form of decision-trees learned from existing ratings (Sun et al., 2013b). An extensive review can be found in Elahi et al. (2016). Intuitively, our model learns a compact, parametric representation of a decision tree end-to-end, by directly maximizing task performance. We evaluate our active learner on MovieLens-20M, a standard dataset for recommendation tasks.

We provide hints to our model during training using samples from an oracle policy that knows all the labels. Related approaches have been explored in previous work on imitation learning and learning to search (Ross & Bagnell, 2014; Chang et al., 2015). These methods, which focus the cost of sampling from the oracle policy on states visited by the model policy, have recently been adopted by researchers working with deep networks for representation learning (Zhang & Cho, 2017; Sun et al., 2017).

3. Model Description

We now present our model, which *metalearns* algorithms for active learning. Our model metalearns by attempting to actively learn on tasks sampled from a distribution over tasks, using supervised feedback to improve its expected performance on new tasks drawn from a similar distribution. Succinctly, our model solves each task by adaptively selecting items for the labeled support set used by a Matching Network (Vinyals et al., 2016) to classify test items. The full support set from which our model selects these examples contains both labeled and unlabeled data.

For a summary of our model, see the architecture diagram in Figure 1, the optimization objectives in Equations 2, 3, and 5, and the pseudo-code in Algorithm 1. We present a formal description of our meta active learning task in Section 3.1. We describe the details of our model in Section 3.2, and our approach to parameter optimization in Section 3.3.

3.1. Task Description

Our model refines its behaviour over many *training episodes*, in order to maximize performance during *test episodes* not encountered in training. In each episode, our model interacts with a support set $S \equiv \{(x, y)\}$ comprising items x for which the model can request labels y, and a similarly-defined evaluation set $E \equiv \{(\hat{x}, \hat{y})\}$. Let $S_t^u \equiv \{(x, \cdot)\}$

denote the set of items in the support set whose labels are still unknown after t label queries, and let $S_t^k \equiv \{(x, y)\}$ denote the complementary set of items whose labels are known. Let S_t denote the joint set of labeled and unlabeled items after t label queries. Let the real-valued vector s_t denote the *control state* of our model after viewing t labels, and let $R(E, S_t, s_t)$ denote the reward won by our model when predicting labels for the evaluation set based on the information it has received after t label queries. We assume all functions depend on the model parameters θ, and omit this dependence from our notation for brevity.

We define the prediction reward as follows:

$$R(E, S_t, s_t) \equiv \sum_{(\hat{x}, \hat{y}) \in E} \log p(\hat{y} | \hat{x}, s_t, S_t), \qquad (1)$$

which gives log-likelihood of the predictions $p(\hat{y} | \hat{x}, s_t, S_t)$ on the evaluation set. The prediction $\hat{y}$ conditions on: the test item $\hat{x}$, the current control state s_t, and the current labeled/unlabeled support set S_t. For tests on Omniglot (see Section 4.1), we use negative cross-entropy on the class labels, and for MovieLens (see Section 4.2), we use the negative Root Mean Squared Error (RMSE).

At each step t of active learning, the model requests the label for an item x from the set S_{t-1}^u and updates its control state from s_{t-1} to s_t based on the response. Together, s_t and S_t determine the model's predictions for test items and the model's decision about which label to request next. Algorithm 1 describes this process in detail, and Section 3.2 formally describes the functions used in Algorithm 1.

The idealized objective for training our model is:

$$\underset{\theta}{\text{maximize}} \; \underset{(S, E) \sim \mathcal{D}}{\mathbb{E}} \left[\underset{\pi(S, T)}{\mathbb{E}} \left[\sum_{t=1}^{T} R(E, S_t, s_t) \right] \right], \quad (2)$$

in which T is the max number of label queries to perform, (S, E) indicates an episode sampled from some distribution $\mathcal{D}$, and $\pi(S, T)$ indicates unrolling the model's active learning policy π for T steps on support set S. Unrolling π produces the intermediate states $\{(S_1, s_1), ..., (S_T, s_T)\}$.

To optimize this objective, our model repeatedly samples an episode (S, E), then unrolls π for T steps of active learning, and maximizes the prediction reward $R(E, S_t, s_t)$ at each step. Alternately, our model could maximize only the reward at the final step. We maximize reward at each step in order to promote *anytime* behaviour – i.e. the model should perform as well as possible after each label query. Anytime behaviour is desirable in many practical settings, e.g. for movie recommendations the model should be robust to early termination while eliciting the preferences of a new user.

During training, for computational efficiency, our model

maximizes the following approximation of Equation 2:

$$\underset{(S, E) \sim \mathcal{D}}{\mathbb{E}} \left[\underset{\pi(S, T)}{\mathbb{E}} \left[\sum_{t=1}^{T} \tilde{R}(S_t^u, S_t, s_t) + R(E, S_T, s_T) \right] \right],$$
$$(3)$$

in which $\tilde{R}(S_t^u, S_t, s_t)$ is a prediction reward for unlabeled items in the support set. We assume labels for the full support set are available during training. We compute $\tilde{R}(S_t^u, S_t, s_t)$ using a *fast prediction* module, and compute $R(E, S_t, s_t)$ using a *slow prediction* module. The fast and slow prediction rewards can be obtained by substituting the appropriate predictions into Equation 1. Sections 3.2.5 and 3.2.6 describe these modules in detail.

3.2. Model Architecture Details

Our model comprises multiple modules: *context-free* and *context-sensitive* encoding, *controller*, *selection*, *reading*, *fast prediction*, and *slow prediction*. We present an overview of our model in Fig. 1 and Alg. 1, which describe how our model's modules perform active learning. The rest of this subsection describes the individual modules in more detail.

Algorithm 1 End-to-end active learning loop (for Eq. 3)

1: *# encode items in S with context-sensitive encoder*
2: *# and encode items in E with context-free encoder*
3: $S = \{(x, y)\}, S_0^u = \{(x, \cdot)\}, S_0^k = \emptyset, E = \{(\hat{x}, \hat{y})\}$
4: **for** $t = 1 \ldots T$ **do**
5:     *# select next instance*
6:     $i \leftarrow \text{SELECT}(S_{t-1}^u, S_{t-1}^k, s_{t-1})$
7:     *# read labeled instance and update controller*
8:     $(x_i, y_i) \leftarrow \text{READ}(S, i)$
9:     $s_t \leftarrow \text{UPDATE}(s_{t-1}, x_i, y_i)$
10:     *# update known / unknown set*
11:     $S_t^k \leftarrow S_{t-1}^k \cup \{(x_i, y_i)\}$
12:     $S_t^u \leftarrow S_{t-1}^u \setminus \{(x_i, \cdot)\}$
13:     *# perform fast prediction*
14:     $L_t^S \leftarrow \text{FAST-PRED}(S, S_t^u, S_t^k, s_t)$
15: **end for**
16: *# perform slow prediction*
17: $L_T^E \leftarrow \text{SLOW-PRED}(E, S_T^u, S_T^k, s_T)$

3.2.1. CONTEXT-[FREE|SENSITIVE] ENCODING

The context-free encoder associates each item with an embedding independent of the context in which the item was presented. For our Omniglot tests, this encoder is a convnet with two convolutional layers that downsample via strided convolution, followed by another convolutional layer and a fully-connected linear layer which produces the final context-free embedding. For our MovieLens tests, this encoder is a simple look-up-table mapping movie ids to
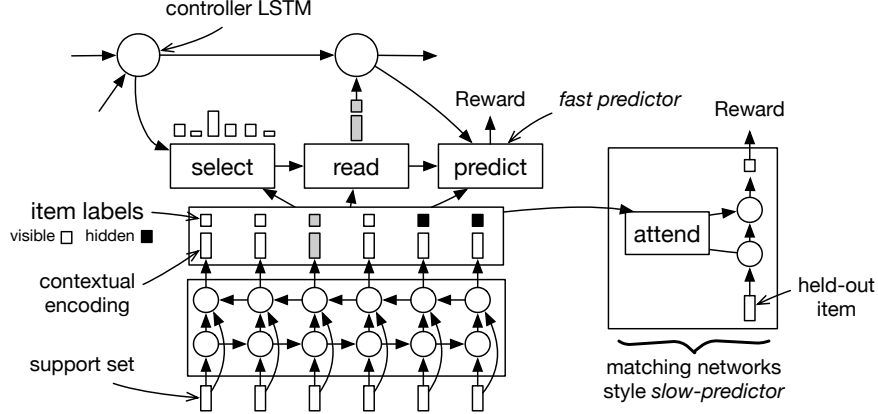
Figure 1. A summary of the modules in our model. Items in the support and evaluation set are embedded using a context-free encoder. Final embeddings for support set items are computed by processing their context-free embeddings with a context-sensitive encoder. The selection module places a distribution over unlabelled items in S_t^u using a gated combination of controller-item similarity features and item-item similarity features. The read module copies the selected item and its label, and transforms them for input to the controller, which then updates its state from s_{t-1} to s_t. Fast predictions are made within the support set S based on sharpened item-item similarity features. Slow predictions are made for items in the held-out set E using a Matching Network-style function which incorporates masking to account for known/unknown labels, and conditions on the state s_t. We train this system end-to-end with Reinforcement Learning.

embeddings. We denote the context-free encoding of item $x_i \in S$ as x_i', and similarly define $\hat{x}_i'$ for $\hat{x}_i \in E$.

The context-sensitive encoder produces an embedding x_i'' for each item $x_i \in S$ based on the context-free embeddings $x_j' : \forall x_j \in S$. The context-sensitive encoder is not applied to items in the evaluation set. Our model uses a modified form of the encoder from Matching Networks (Vinyals et al., 2016). Specifically, we run a bidirectional LSTM (Hochreiter & Schmidhuber, 1997; Schuster & Paliwal, 1997) over all context-free embeddings for the support set, and then add a linear function of the concatenated forwards and backwards states to the context-free embedding x_i' to get x_i''.

We can write this as follows:

$$x_i'' = x_i' + W_e \left[\vec{h}_i; \overleftarrow{h}_i \right], \tag{4}$$

in which $\vec{h}_i$ gives the forward encoder state for item x_i, $\overleftarrow{h}_i$ gives the backward encoder state, and W_e is a trainable matrix. We compute the forward states $\vec{h}_i$ as in a standard LSTM, processing the support set items x_i sequentially following a random order. We compute the backward states $\overleftarrow{h}_i$ by processing the sequence of concatenated x_i' and $\vec{h}_i$ vectors in reverse.

3.2.2. READING

This module concatenates the embedding x_i'' and label y_i for the item indicated by the selection module, and linearly transforms them before passing them to the controller (Alg. 1, line 8).

3.2.3. CONTROLLER

At each step t, the controller receives an input r_t from the reading module which encodes the most recently read item/label pair. Additional inputs could take advantage of task-specific information. The control module performs an LSTM update:

$$s_t = \text{LSTM}(s_{t-1}, r_t).$$

We initialize s_0 for each episode (S, E) using the final state of the backwards LSTM in the context-sensitive encoder which processed the support set S. In principal, this allows the controller to condition its behaviour on the full unlabeled contents of the support set (Alg. 1, line 9).

3.2.4. SELECTION

At each step t, the selection module places a distribution P_t^u over all unlabeled items $x_t^u \in S_t^u$. It then samples the index of an item to label from P_t^u, and feeds it to the reading module (Alg. 1, line 6).

Our model computes P_t^u using a gated, linear combination of features which measure controller-item similarity and item-item similarity. For each item, we compute the controller-item similarity features:

$$b_t^i = x_i'' \odot W_b s_t,$$

where W_b is a trainable matrix and $\odot$ indicates elementwise multiplication. We also compute the following six item-item similarity features: [max|mean|min] cosine similarity to any labeled item, and [max|mean|min] cosine similarity to any unlabeled item. We concatenate the controller-item

similarity features and item-item similarity features to get a vector d_t^i. We also compute a gating vector: $g_t = \sigma(W_g s_t)$, in which W_g is a trainable matrix and $\sigma(\cdot)$ indicates the standard sigmoid.

For each $x_i^u \in S_t^u$, we compute the selection logit:

$$p_t^i = (g_t \odot d_t^i)^\top w_p,$$

where w_p indicates a trainable vector. Finally, we compute P_t^u by normalizing over the logits $p_t^i : \forall x_i^u \in S_t^u$ via softmax. This module performs worse when the controller-item or item-item features are removed. Our model intelligently adapts these heuristics to the task at hand. We provide pseudo-code for how these modules interact during active learning in Algorithm 1.

3.2.5. FAST PREDICTION

The fast prediction module makes an attention-based prediction for each unlabeled item $x_i^u \in S_t^u$ using its cosine similarities to the labeled items $x_j^k \in S_t^k$, which are sharpened by a non-negative matching score γ_t^i between x_i^u and the control state s_t. The cosine similarities are taken between the context-sensitive embeddings x_i'' and x_j'' of the respective items. These do not change with t and may be precomputed before unrolling the active learning policy. Predictions from this module are thus fast to compute while unrolling the policy (Alg. 1, line 14). The precomputed cosine similarities may be reused in the selection module for computing item-item similarity features, further amortizing their cost.

For each unlabeled x_i^u, we compute a set of attention weights over the labeled $x_j^k \in S_t^k$ by applying a softmax to the relevant cosine similarities, using γ_t^i as a temperature for the softmax. We compute the sharpening term as follows:

$$\gamma_t^i = \exp((x_i'')^\top W_\gamma s_t),$$

where W_γ indicates a trainable matrix. This module performs significantly worse without the sharpening term. The final *fast prediction* is formed by taking a convex combination of the labels y_j for the labeled $x_j^k \in S_t^k$ using the computed attention weights.

3.2.6. SLOW PREDICTION

The slow prediction module implements a modified Matching Network prediction, which accounts for the distinction between labeled and unlabeled items in S_t, and conditions on the active learning control state s_t (Alg. 1, line 17).

Given the context-free embedding $\hat{x}'$ for some held-out example $\hat{x} \in E$, the state s_t, and required initial values, this module predicts a label by iterating the steps:

1. $m_k = \text{LSTM}(m_{k-1}, \tilde{x}_{k-1}, \hat{x}', s_t)$

2. $\hat{x}'' = \hat{x}' + W_m m_k$
3. $\tilde{a}_k = \text{attend}(\hat{x}'', S_t^k)$
4. $\tilde{x}_k, \tilde{y}_k = \text{attRead}(\tilde{a}_k, S_t^k)$

Here, LSTM is an LSTM update, m_k is the *matching state* at step k, $\hat{x}''$ is the match-sensitive embedding of $\hat{x}$ at step k, W_m is a trainable matrix, $\tilde{a}_k$ are the matching attention weights at step k, $\tilde{x}_k$ is the "item attention" result from step k, and $\tilde{y}_k$ is the "label attention" result from step k. For details of the attend and attRead functions, refer to Vinyals et al. (2016). As a final prediction, this module returns the label attention result $\tilde{y}_K$ from the Kth (final) step of iterative matching. In our tests we fix $K = 3$.

Note: our model contains many linear transforms Wv. Our model adds bias and gain terms to all of these transforms, as described for weight normalization (Salimans & Kingma, 2016). We omit these terms for brevity. Similarly, we use layer normalization in our active learning and matching controller LSTMs (Ba et al., 2016).

3.3. Training the Model

We optimize the parameters of our model using a combination of backpropagation and policy gradients. For a clear review of optimization techniques for general stochastic computation graphs, see Schulman et al. (2016a).

Using the notation from Section 3.1 and following the approach of (Schulman et al., 2016a), we can write the gradient of our training objective as follows:

$$\nabla_\theta R(S, E, \theta) = \qquad (5)$$
$$\mathbb{E}_{p(\vec{S}|(S,E))} \left(\nabla_\theta \log p(\vec{S}|(S, E)) \left[R(\vec{S}) \right] + \nabla_\theta R(\vec{S}) \right),$$

in which $R(S, E, \theta)$ denotes the expected reward won by the active learning model while working on episode (S, E). $\vec{S}$ denotes the set of intermediate states $\{(S_t, s_t)\}$ generated by the model while working with (S, E). $R(\vec{S})$ denotes the sum of rewards (as described in Equation 3) received by the model while working on episode (S, E). In the term $\nabla_\theta R(\vec{S})$, all decisions made by the model to produce $\vec{S}$ are treated as constant. Taking the expectation of Equation 5 with respect to episodes $(S, E) \sim \mathcal{D}$ gives us an unbiased gradient estimator for the objective in Equation 3.

Rather than using the gradients in Equation 5 directly, we optimize the model parameters using Generalized Advantage Estimation (Schulman et al., 2016b), which provides an actor-critic approach for approximately optimizing the policy gradients in Equation 5. For more details on how Generalized Advantage Estimation helps reach a favourable bias-variance trade-off in policy gradient estimation, see the source paper (Schulman et al., 2016b). We apply the GAE updates using ADAM (Kingma & Ba, 2015).
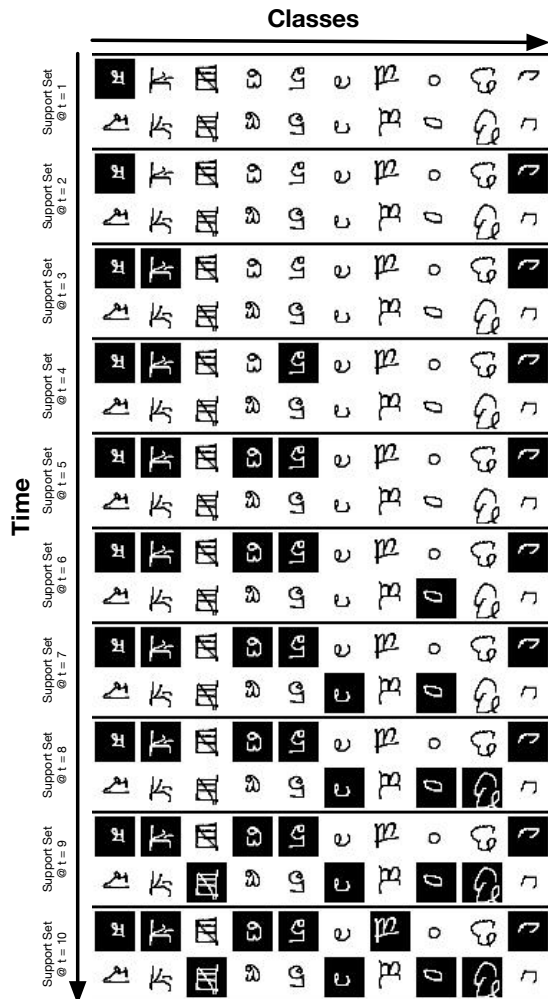
Classes

Figure 2. A rollout of our active learning policy for Omniglot, using a support set of 20 items from 10 different classes with 2 items per class. Each row represents the support set at different active iterations. For visualization purposes, each column represents a class. Unlabeled items have white background while selected items have black background. In this case, the model behaves optimally by selecting at each step an item with a yet-unseen label.

4. Experiments

4.1. Omniglot

We run our first experiments on the Omniglot dataset (Lake et al., 2015) consisting of 1623 characters from 50 different alphabets each hand-written by 20 different persons. Following Vinyals et al. (2016), we divide the dataset into 1200 characters for training and keep the rest for testing. When measuring test performance, our model interacts with characters it did not encounter during training.

For the context-free embedding function we use a three-layer convolutional network. The first two layers use 5×5 convolutions with 64 filters and downsample with a double stride. The third layer uses a 3×3 convolution with 64

filters and no downsampling. These layers produce a $7 \times 7 \times 64$ feature map that we flatten and pass through a fully connected layer. All convolutional layers use the leaky ReLU nonlinearity (Maas et al., 2013).

We setup N-way, K-shot Omniglot classification as follows. We randomly pick N character classes from the available train/test classes. Then, we build a support set by randomly sampling 5 items for each character class, e.g. in the 5-way setting, there are 25-items in the support set. The held-out set is always obtained by randomly sampling 1 item per class. In our active learning setting, K-shot is proportional to how many labels the model can acquire. In the N-way, K-shot setting, the model asks for NK labels before performing held-out prediction. For example, in 5-way, 1-shot classification, the model asks for 5 labels. Following each label query, we also measure anytime performance of the fast prediction module on the items remaining in S_t^u. Note that the 1-shot setting is particularly challenging for our model, as it needs to ask for different classes at each step, and the ability to identify missing classes is limited by the accuracy of the underlying one-shot classifier.

We compare our active learner to four baselines. To compute a pessimistic estimate of potential performance, we use a matching network where we label NK items chosen at random from the full support set (Matching Net (random)). As the labels are randomly sampled, it is possible that a given class is never represented among the labeled items and the model cannot classify perfectly, even in principle. To compute an optimistic estimate of potential performance, we measure the "ideal" matching network accuracy by labeling a class-balanced subset of items from the full support set (Matching Net (balanced)). This baseline represents a highly-performant policy that the active learner can, in principle, learn. For the last baseline (Min-Max-Cos), we formulate a heuristic policy. At each active learning step, we select the item which has minimum maximum cosine similarity to unlabeled items in the support set. This heuristic selects item that are different from each other, a strategy well-suited to the Omniglot classification task where items are drawn from a consistent set of underlying classes.

We report the results in Table 1. Matching Networks operating on a randomly sampled set of labels suffer the most in 1-shot scenarios, where the probability of all classes being represented is particularly low (especially in the 10-way case). Overall, the active policy nearly matches the performance of the optimistic balanced Matching Network baseline. Degradation of performance by 2.2% is observed for the 1-shot, 10-way case. This is not surprising since augmenting the number of classes in the support set, while keeping the number of shots fixed, considerably increases the difficulty of the problem for the active learner. Figure 2 shows a roll-out of the model policy in the 10-way setting.

Table 1. Results for our active learner and baselines for the N-way, K-shot classification settings.

Model	5-way			10-way		
	1-shot	2-shot	3-shot	1-shot	2-shot	3-shot
Matching Net (random)	$69.8\%_{\pm 0.10}$	$93.1\%_{\pm 0.07}$	$98.5\%_{\pm 0.04}$	$67.3\%_{\pm 0.10}$	$91.2\%_{\pm 0.06}$	$97.6\%_{\pm 0.06}$
Matching Net (balanced)	$97.9\%_{\pm 0.07}$	$98.9\%_{\pm 0.07}$	$99.2\%_{\pm 0.06}$	$96.5\%_{\pm 0.04}$	$98.3\%_{\pm 0.03}$	$98.7\%_{\pm 0.05}$
Active MN	$97.4\%_{\pm 0.11}$	$99.0\%_{\pm 0.08}$	$99.3\%_{\pm 0.03}$	$94.3\%_{\pm 0.24}$	$98.0\%_{\pm 0.07}$	$98.5\%_{\pm 0.06}$
Min-Max-Cos	$97.4\%_{\pm 0.11}$	$99.3\%_{\pm 0.02}$	$99.4\%_{\pm 0.04}$	$93.5\%_{\pm 0.11}$	$98.4\%_{\pm 0.02}$	$98.8\%_{\pm 0.03}$

Figure 3 provides results for the more challenging setting of 20-way classification. We tested two properties of our model: its anytime performance beyond the 1-shot setting, and its ability to generalize to problems with more classes than were seen during training. The model performed well on 20-way classification, and quickly approached the optimistic estimate after acquiring more labels. We also found that policies trained for as little as 10-way classification could generalize to the 20-way setting.

Our model relies on a number of moving parts. When designing the architecture, we followed the simple approach of minimizing changes to the original Matching Network from Vinyals et al. (2016). We now provide ablation test results for several parts of our model. In the 10-way, 1-shot setting accuracy dropped from 94.5 to 86.0 when we removed attention temperature from the fast prediction module. Reducing the number of matching steps from 3 to 2 or 1 had no significant effect in this setting. Removing the context-sensitive encoder also had no significant effect. Streamlining our architecture is clearly a useful topic for future work aimed at scaling our approach to more realistic settings.

4.2. MovieLens

4.2.1. Setup

We test our model in the "cold-start" collaborative filtering scenario using the publicly available MovieLens-20M dataset.[1] The dataset contains approximately 20M ratings on 27K movies by 138K users. The ratings are on an ordinal 10-point scale, from 0.5 to 5 with intervals of 0.5. We subsample the dataset by selecting 4000 movies and 6000 users with the most ratings. After filtering, the dataset contains approximately 1M ratings. We partition the data randomly into 5000 training users and 1000 test users. The training set represents the users already in the system who are used to fit the model parameters. We use the test users to evaluate our active learning approach. For each user, we randomly pick 50 ratings to include in the support set (movies that the user can be queried about) and 10 movies and ratings for the held-out set. We ensure that movies in the held-out set and

---

[1]Available at http://grouplens.org/datasets/movielens/

in the support set do not overlap. We train our active learner to minimize the mean-squared error (MSE) with respect to the true rating. We adapt the prediction modules of our model to output the rating for a held-out item as follows: we compute a convex combination of the ratings of "visible" movies in the support set (the movies the active learner has already queried about), where the weights are given by the final attention step of the slow predictor. Although more complex strategies are possible, we empirically found this simple strategy to work well in our experiments. For evaluation, we sample 25000 episodes (i.e. 50 support ratings and 10 held-out ratings from the same user) from the test set and we compute the average per-user root mean-squared error (RMSE), i.e. we average the ratings in the held-out set of each test episode and then average across test episodes. We report the average performance obtained by 3 runs with different random seeds.

4.2.2. Movie Embeddings

For each movie, we pretrain an embedding vector by decomposing the full user/movie rating matrix using a latent factor model (Koren, 2010). This process only uses the training set. For each user u and movie m, we estimate the true rating $r_{u,m}$ with a linear model $\hat{r}_{u,m} = x_u^\top x_m + b_u + b_m + \beta$, where x_u, x_m are the user and movie embedding respectively and b_u, b_m, β are the user, movie, and global bias, respectively. We train the latent factor model by minimizing the mean squared error between true rating r and predicted rating $\hat{r}$. We use the trained x_m as input representations for the movies throughout our experiments.

4.2.3. Results

In Figure 4 we report the results of our active model against various baselines. The Regression baseline performs regularized linear regression on movies from the support set whose ratings have been observed incrementally in random order. Because of the small amount of training data, for each additional label we tune the regularization parameter by monitoring performance on a separate set of validation episodes. The Gaussian Process baseline selects the next movie to label in proportion to the variance of the predictive posterior distribution over its rating. This gives an
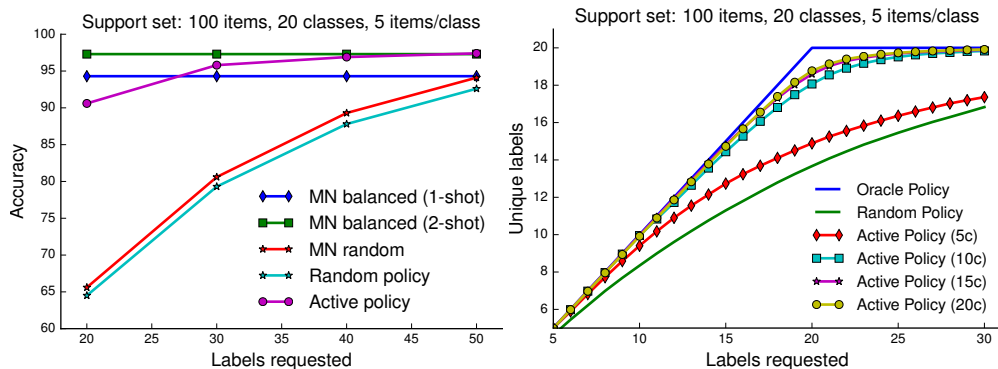
Figure 3. Experiment results for our model and baselines on Omniglot. The left plot shows how prediction accuracy improves with the number of labels requested in a challenging 20-way setting. After 20 label requests (corresponding to a 20-way, 1-shot problem), the active policy outperforms random policy and random MN baselines, but is inferior to the balanced MN. After 30 labels, the active policy nearly matches the performance of the balanced MN using 40 labels (20-way, 2-shot). The right plot shows the number of unique labels with respect to the number of requested labels for models trained on problems with 5-20 classes, and tested on 20-way classification. This gives an idea of how models search for labels from unseen groups and generalize to problems with different numbers of classes.
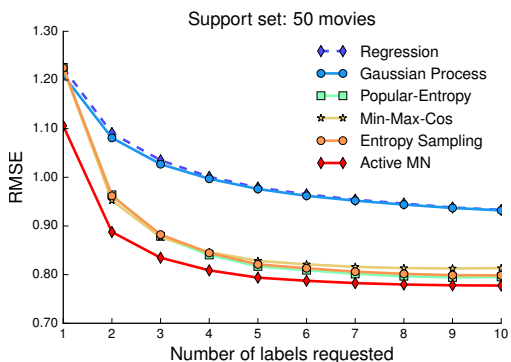

Figure 4. Performance of the model and baselines measured with RMSE on the Movielens dataset.

idea of the impact of using MN one-shot capabilities rather than standard regression techniques. The Popular-Entropy, Min-Max-Cos, Entropy Sampling baselines train our model end-to-end, but using a fixed selection policy. Specifically, we train our architecture end-to-end, but instead of training an active learning policy through the select module we choose items from the support set incrementally according to a heuristic policy. This gives an idea of the importance of learning the selection policy. The Popular-Entropy policy, adapted from the cold-start work of Rashid et al. (2002), a priori scores each item in the support set according to the logarithm of its popularity multiplied by the entropy of the item's ratings measured across users. This strategy aims to collect first the ratings for those movies that are both popular and have been rated differently by different users. Although it is simplistic, the policy achieves competitive performance for bootstrapping a system from a cold-start setting (Elahi et al., 2016). The Min-Max-Cos policy is identical to the synonymous baseline used for Omniglot, i.e. it selects the unrated movie which has minimum maximum cosine similarity to any of the rated movies. Roughly, this

selects the unrated movie which differs most from the rated movies. Entropy Sampling selects movies in proportion to rating prediction entropy.

The active policy learned end-to-end outperforms the baselines in terms of RMSE, particularly after requesting only the first few labels. After 10 ratings, our model achieves an improvement of 2.5% in RMSE against the best baseline. Unsurprisingly, the gap diminishes with a higher number of labels requested. After observing 5 labels, the Popular-Entropy baseline and our architecture equipped with the Min-Max-Cos heuristic converge toward the active policy but never quite match it. For MovieLens, where labels are user-dependent and not tied to an underlying class, a data-driven selection policy may adapt better to the task. This contrasts with the Omniglot setting, where there is no aspect of personalization and Active MN and Min-Max-Cos perform similarly. The heuristic is designed not to select items similar to those it has already seen, but doing so can be beneficial in personalized settings (Elahi et al., 2016).

5. Conclusion

We introduced a model that learns active learning algorithms end-to-end. Our goal was to move away from engineered selection heuristics towards strategies learned directly from data. Our model leverages labeled instances from different but related tasks to learn a selection strategy for the task at hand, while simultaneously adapting its representation of the data and its prediction function. We evaluated the model on "active" variants of one-shot learning tasks for Omniglot, demonstrating that its policy approaches an optimistic performance estimate. On a cold-start collaborative filtering task derived from MovieLens, the model outperforms several baselines and shows promise for application in more realistic settings.

References

Aggarwal, Charu C. *Recommender Systems*. Springer, 2016.

Ba, Jimmy Lei, Kiros, Jamie Ryan, and Hinton, Geoffrey E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Chang, Kai-Wei, Krishnamurthy, Akshay, Agarwal, Alekh, III, Hal Daumé, and Langford, John. Learning to search better than your teacher. *International Conference on Machine Learning (ICML)*, 2015.

Cohn, David A, Ghahramani, Zoubin, and Jordan, Michael I. Active learning with statistical models. *Journal of artificial intelligence research*, 4(1):129–145, 1996.

Elahi, Mehdi, Ricci, Francesco, and Rubens, Neil. A survey of active learning in collaborative filtering recommender systems. *Computer Science Review*, 20:29–50, 2016.

Gilad-Bachrach, Ran, Navot, Amir, and Tishby, Naftali. Query by committee made real. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*, pp. 443–450. MIT Press, 2005.

Golbandi, Nadav, Koren, Yehuda, and Lempel, Ronny. On bootstrapping recommender systems. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, 2010.

Golbandi, Nadav, Koren, Yehuda, and Lempel, Ronny. Adaptive bootstrapping of recommender systems using decision trees. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, 2011.

Hannun, Awni, Case, Carl, Casper, Jared, Catanzaro, Bryan, Diamos, Greg, Elsen, Erich, Prenger, Ryan, Satheesh, Sanjeev, Sengupta, Shubho, Coates, Adam, and Ng, Andrew Y. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567v2*, 2014.

Harpale, Abhay S and Yang, Yiming. Personalized active learning for collaborative filtering. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 91–98. ACM, 2008.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural Computation*, 9.8:1735–1780, 1997.

Hoi, Steven CH, Jin, Rong, Zhu, Jianke, and Lyu, Michael R. Batch mode active learning and its application to medical image classification. In *Proceedings of the 23rd international conference on Machine learning*, pp. 417–424. ACM, 2006.

Houlsby, Neil, Huszár, Ferenc, Ghahramani, Zoubin, and Lengyel, Máté. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Houlsby, Neil, Hernandez, Jose, and Ghahramani, Zoubin. Coldstart active learning with robust ordinal matrix factorization. In *Proceedings of The 31st International Conference on Machine Learning*, pp. 766–774, 2014.

Kawale, Jaya, Bui, Hung H, Kveton, Branislav, Tran-Thanh, Long, and Chawla, Sanjay. Efficient thompson sampling for online matrix-factorization recommendation. In *Advances in Neural Information Processing Systems (NIPS)*. 2015.

Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv:1412.6980v2 [cs.LG]*, 2015.

Koch, Gregory. *Siamese neural networks for one-shot image recognition*. PhD thesis, University of Toronto, 2015.

Koren, Yehuda. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):1, 2010.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Lake, Brenden M, Salakhutdinov, Ruslan, and Tenenbaum, Joshua B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Lewis, David D and Gale, William A. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 3–12. Springer-Verlag New York, Inc., 1994.

Lika, Blerina, Kolomvatsos, Kostas, and Hadjiefthymiades, Stathes. Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4):2065–2073, 2014.

Maas, Andrew L, Hannun, Awni Y, and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. *International Conference on Machine Learning (ICML)*, 2013.

Rashid, Al Mamunur, Albert, Istvan, Cosley, Dan, Lam, Shyong K, McNee, Sean M, Konstan, Joseph A, and Riedl, John. Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pp. 127–134. ACM, 2002.

Rashid, Al Mamunur, Karypis, George, and Riedl, John. Learning preferences of new users in recommender systems: an information theoretic approach. *ACM SIGKDD Explorations Newsletter*, 10(2):90–100, 2008.

Ross, Stéphane and Bagnell, J. Andrew. Reinforcement and imitation learning via interactive no-regret learning. *arXiv:14065979v1 [cs.LG]*, 2014.

Salimans, Tim and Kingma, Diederik P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*. 2016.

Santoro, Adam, Bartunov, Sergey, Botvinick, Matthew, Wierstra, Daan, and Lillicrap, Timothy. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

Schölkopf, Bernhard and Smola, Alexander J. Learning with kernels: support vector machines, regularization, optimization, and beyond, 2002.

Schulman, John, Heess, Nicolas, Weber, Theophane, and Abbeel, Pieter. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems (NIPS)*, 2016a.

Schulman, John, Moritz, Philipp, Levine, Sergey, Jordan, Michael I, and Abbeel, Pieter. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016b.

Schuster, M. and Paliwal, K.K. Bidirectional recurrent neural networks. *Transactions on Signal Processing*, 45(11), 1997.

Settles, Burr. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

Sun, Mingxuan, Li, Fuxin, Lee, Joonseok, Zhou, Ke, Lebanon, Guy, and Zha, Hongyuan. Learning multiple-question decision trees for cold-start recommendation. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, 2013a.

Sun, Mingxuan, Li, Fuxin, Lee, Joonseok, Zhou, Ke, Lebanon, Guy, and Zha, Hongyuan. Learning multiple-question decision trees for cold-start recommendation. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pp. 445–454. ACM, 2013b.

Sun, Wen, Venkatraman, Arun, Gordon, Geoffrey J, Boots, Byron, and Bagnell, J Andrew. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *International Conference on Machine Learning (ICML)*, 2017.

Tong, Simon and Chang, Edward. Support vector machine active learning for image retrieval. In *Proceedings of the ninth ACM international conference on Multimedia*, pp. 107–118. ACM, 2001.

Vinyals, Oriol, Blundell, Charles, Lillicrap, Tim, Wierstra, Daan, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3630–3638, 2016.

Woodward, Mark and Finn, Chelsea. Active one-shot learning. In *NIPS Workshop*, 2016.

Wu, Yonghui, Schuster, Mike, Chen, Zhifeng, Le, Quoc V., Norouzi, Mohammad, Macherey, Wolfgang, Krikun, Maxim, Cao, Yuan, Gao, Qin, Macherey, Klaus, Klingner, Jeff, Shah, Apurva, Johnson, Melvin, Liu, Xiaobing, Kaiser, Lukasz, Gouws, Stephan, Kato, Yoshikiyo, Kudo, Taku, Kazawa, Hideto, Stevens, Keith, Kurian, George, Patil, Nishant, Wang, Wei, Young, Cliff, Smith, Jason, Riesa, Jason, Rudnick, Alex, Vinyals, Oriol, Corrado, Greg, Hughes, Macduff, and Dean, Jeffrey. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Zhang, Jiakai and Cho, Kyunghyun. Query-efficient imitation learning for end-to-end autonomous driving. *American Association for Artificial Intelligence (AAAI)*, 2017.

Zilberstein, Shlomo. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73, 1996.