Learning a Better Negative Sampling Policy with Deep Neural Networks for Search

Daniel Cohen
Center for Intelligent Information Retrieval
University of Massachusetts Amherst
dcohen@cs.umass.edu

Scott M. Jordan
Autonomous Learning Laboratory
University of Massachusetts Amherst
sjordan@cs.umass.edu

W. Bruce Croft
Center for Intelligent Information Retrieval
University of Massachusetts Amherst
croft@cs.umass.edu

ABSTRACT

In information retrieval, sampling methods used to select documents for neural models must often deal with large class imbalances during training. This issue necessitates careful selection of negative instances when training neural models to avoid the risk of overfitting. For most work, heuristic sampling approaches, or policies, are created based off of domain experts, such as choosing samples with high BM25 scores or a random process over candidate documents. However, these sampling approaches are done with the test distribution in mind. In this paper, we demonstrate that the method chosen to sample negative documents during training plays a critical role in both the stability of training, as well as overall performance. Furthermore, we establish that using reinforcement learning to optimize a policy over a set of sampling functions can significantly improve performance over standard training practices with respect to IR metrics and is robust to hyperparameters and random seeds.

ACM Reference Format:
Daniel Cohen, Scott M. Jordan, and W. Bruce Croft. 2019. Learning a Better Negative Sampling Policy with Deep Neural Networks for Search. In *The 2019 ACM SIGIR International Conference on the Theory of Information Retrieval (ICTIR '19), October 2–5, 2019, Santa Clara, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3341981.3344220

1 INTRODUCTION

The recent application of deep learning techniques has facilitated advancements in the field of Information Retrieval (IR) [1, 8, 18]; however, the recent surge in performance for some search tasks has overlooked significant issues that impact these models. As these current deep neural models rely on many parameters, large collections are required to simultaneously learn an effective representation and relevance function through a stochastic optimization process. IR, in particular, exacerbates this problem, as while collections are often large in size, the number of judged documents, both relevant and non-relevant, is small. As the majority of non judged documents are non-relevant, this disparity in numbers between positive and negative documents results in a significant problem when training any supervised model as discussed in Wang et al. [33]

Two common approaches to handle this imbalance involve oversampling a collection, where the minority class is continually mixed with unseen examples from the majority, is to use either random sampling or external domain knowledge to select documents. However, this is a non trivial task as one cannot train indefinitely on a set number of queries due to overfitting, thus a widespread approach to fully capture the collection is to re-balance the training data through over or under sampling. As discussed by Wang et al. [34], under-sampling, where the majority class is reduced is detrimental to neural models, results in discarding potentially informative samples, and over sampling via increasing the presence of the minority class contributes redundant data at the risk of overfitting.

As neural models often perform a reranking of BM25 or query likelihood (QL) scores[24] as a form of oversampling. These functions are used to generate negative candidate documents that are similar to relevant documents to reduce the drawbacks discussed in the previous paragraph, and for better performance during evaluation. When the final model evaluation is conducted using the same negative sampling method used during training then the result is a positively biased score [19]. This leads to a problem not often addressed in literature, where models are trained knowing how the test distribution will be constructed. Consider the case where an unknown function is to construct candidates for reranking. Would using BM25 to select training documents be an effective policy for training a neural IR model? This would only expose the neural model to a small area of the collection where $tf.idf$ values are high. and potentially fail to generalize when exposed to a random document from a different distribution.

We argue that these sampling policies are critical to training. While selective oversampling results in an effective model for reranking, this biases the model towards performance on a certain area of the collection manifold and does not optimize performance over the whole collection or over a different reranking distribution. A random sampling process used to select documents during training does not bias the model, but this often results in the model learning a general retrieval function that is not effective when documents are similar [19].

We approach the problem of negative sampling from a control perspective, which represents a significant departure from past work [26, 32, 36]. We decompose the training of a neural IR model into two components: an environment, which optimizes the IR model, and an agent, which learns to control the optimization process by selecting documents for the IR model to rank. Within this

ICTIR '19, October 2–5, 2019, Santa Clara, CA, USA

ACM ISBN 978-1-4503-6881-0/19/10…$15.00
https://doi.org/10.1145/3341981.3344220

paradigm, past approaches of choosing a negative document randomly or from a retrieved BM25 list are represented as handcrafted sampling policies.

In this paper, we address the following research questions (RQ):

- **RQ1** To what degree do sampling policies impact performance of neural information retrieval models?
- **RQ2** Can learning a control policy result in a more robust model during training, as well as improve the performance of the model during evaluation?

In addressing these research questions, we empirically establish the impact of the sampling policy on neural IR models. Furthermore, we develop a novel approach that reduces the variance of training neural networks and ensures that the majority of runs perform near the maximum possible.

2 RELATED WORK

Selective negative sampling has been widely used to train models. In the case of natural language processing, noise contrastive estimation [20] as well as hierarchical softmax [12, 13] are used to reduce the vocabulary space when training word embeddings and machine translation models by using common words as negative examples [11, 15]. However, Chen et al. [5] have indicated that using the inner product between words results in a significantly more effective embedding which suggests that there are multiple sampling policies based on the final task.

Our work follows the recent studies of *learning to learn* methods, where a model or distribution is learned that optimizes the performance of another. Zoph and Le as well as Bello et al. propose an approach to remove the heuristic policies humans introduce and allow a learned policy to select network architectures and optimizers [2, 39]. Expanding a policy to include greedy choices, Graves et al. [14] use a bandit approach to select sample difficulty in a curriculum learning environment where a model is presented with samples from separate subcategories of various difficulty levels.

Further driving the importance of meta learning approaches, Fan et al. [9] demonstrate that in a *teacher-student* model, the order of negative documents selected by a policy gradient model results in significantly improved performance and shorter training time for the student model on the image classification task. However, this approach requires uncertainty measurement over the entire collection at each step, which is ineffective for the number of documents typical in IR. Within the realm of text, Wu et al. [37] introduce a a Q-learning based approach to choose the candidate samples to label for the active learning task.

3 METHODS

3.1 Markov Decision Process

We demonstrate that the negative sampling problem can be formalized as a Markov Decision Process [29] via the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, R, d_0, \gamma)$. Here, $\mathcal{S}$ represents the set of possible states the agent can be in, $\mathcal{A}$ is the set of possible actions the agent can select, and $\mathcal{R} \subset (r_{min}, r_{max})$ such that $r_{min} > -\infty, r_{max} < \infty$ is the set of possible rewards that the agent can receive. P is the transition function, $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$, such that $P(s, a, s') := \Pr(S_{t+1} = s' | S_t = s, A_t = a)$ is the transition function that characterizes the distribution over states at time $t+1$ given the state S_t and action A_t at time t. R then represents the reward function that characterizes the distribution $R(s, a, s', r) := \Pr(R_t = r | S_t = s, A_t = a, S_{t+1} = s')$ such that it maps the agent's action between states s, s' to a real value. Lastly, $d_0 := \Pr(S_0 = s)$ represents the initial state distribution, and $\gamma \in [0, 1]$ is the reward discount parameter.

We call the method an agent uses to select an action a policy. A policy, $\pi : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \rightarrow [0, 1]$, is a function parameterized by a weight vector, $\theta \in \mathbb{R}^n$ where $\pi(s, a, \theta) := \Pr(A_t = a | S_t = s, \theta)$. An agent's goal is to approximate a policy that maximizes the expected sum of discounted rewards. This goal is denoted with the objective function, $J(\theta) := \mathbf{E}[G|\theta]$, where $G = \sum_{t=0}^{\infty} \gamma^t R_t$ is called the return and conditioning on θ means actions will be selected according to the policy π using the weights θ. We assume at some finite number of time-steps, T, the agent enters a special state called a terminal absorbing state where all the actions transitions back into this state with probability one and all rewards are zero. The interval of time $t \in [0, T]$ is called an episode and when $t = T$ the episode ends and time is reset to $t = 0$. In the environment used in this paper, an episode represents the training of a neural IR model over multiple epochs until an early stopping condition is met.

3.1.1 Action. $\mathcal{A}$ is a set of retrieval functions, $f : Q \times C \rightarrow C_r$ over the retrieval collection C given queries Q and produces a ranked set C_r. Thus, the agent selects an action within the functional space of the document collection rather than choosing individual documents to sample. In this paper, we restrict this space to two functions, BM25 and a random distribution $d \sim U(C)$. Once the action is selected, an independent process then samples from the set of documents retrieved from this function.

3.1.2 State. $\mathcal{S}$ is a combination of information regarding the IR-Model and the Training Data as shown in Figure 1. Specifically, $s \in \mathcal{S}$ contains two parts: (1) information about the incoming batch with respect to queries and positive documents. (2) information regarding the state of the IR model and training process.

We represent the state set $\mathcal{S}$ as a combination of the current batch and the features of the neural model. The neural retrieval model is represented as the vector

$$< \mathcal{L}(b_{t-1}, \eta_{t-1}), \beta ||\nabla \mathcal{L}(b_{t-1}, \eta_{t-1})||_2, \frac{t}{T_e}, e >$$

where $\mathcal{L}(b_{t-1}, \eta_{t-1})$ is the loss of the network from the previous batch given the network's parameters η_{t-1}, $||\nabla \mathcal{L}(b_{t-1}, \eta_{t-1})||_2$ is the ℓ_2 norm of the gradient in the top n layers of the neural model multiplied by a constant size β. The β parameter is introduced as a scaling factor due to the use of a deep reinforcement learning (RL) agent to bound feature ranges [30]. While neural networks perform best with normalized inputs, this plays a critical role for RL where significant changes in state can result in the distribution over actions collapsing to a single point. As the magnitude of the gradient grows with the number of parameters, β acts as a normalization constant to prevent the agent from collapsing. Lastly, the current step in the epoch, t is normalized with the total number of steps T_e in each epoch, and the current epoch number, e, is included.

As seen in Figure 1, in addition to the above features, the agent also receives a compressed representation of the incoming mini-batch containing information regarding the query and relevant

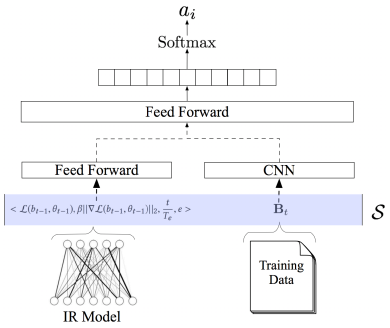

Figure 1: Break down of state information and agent architecture.

document pairs as well as similarity statistics over candidate documents. The compressed representation is done at the query-passage level. For a given sequence $w_1, \ldots, w_n$, we introduce the compression function that represents a weighted term frequency based embedding. Formally, this function, ϕ, leverages an embedding function f_e and term frequency information:

$$\phi(w_1^n) = \sum_i^n f_e(w_i)$$

Each query is represented alongside a relevant document in the matrix $\mathbf{B} \in \mathbb{R}^{2|b| \times d}$ such that

$$\mathbf{B} = [\phi(Q_1), \phi(D_{1_r}), \ldots, \phi(Q_{|b|}), \phi(D_{|b|_r})]^\intercal$$

where $|b|$ is the number of queries in the minibatch and d_e is the embedding size of ϕ. The advantage of structuring the batch in this representation allows for a series of wide convolutions to be pulled to learn a distributed representation of the state with respect to the query and positive examples. A feedforward approach was experimented with, but failed to yield performance better than random.

3.1.3 Reward. As discussed by Ng et al. [22], seemingly intuitive reward functions result in drastically different behaviour than what was expected. While reward shaping is still an active field of research, the authors suggest to only reward for the event you want the agent to learn to maximize. We want the agent to learn to maximize the IR model's performance on the validation set during training. To this end, we consider the following three reward functions: $\mathcal{R}_{raw}, \mathcal{R}_{ceil}$ and $\mathcal{R}_{diff}$. The function R_{raw} rewards the agent using the mean average precision (MAP) score on held out query set, i.e., $R_t = MAP(q, \eta_t)$, where MAP is a function that computes the MAP score on a set of queries q, using an IR model with weights η_t. For this reward function the agent needs to maximize the sum of MAP scores the IR models can achieve. The function R_{ceil} rewards the agent using the difference of the maximum possible MAP score on the collection, $m_{\max}$, and the current MAP score, i.e., $R_t = MAP(q, \eta_t) - m_{\max}$. With this reward function the agent tries to maximize the sum of rewards, which is non-positive. The hope is the agent will learn to minimize the time it takes to converge the IR model. The last reward function we define, R_{diff}, is computed from the change in the IR model's performance, i.e., $R_t = MAP(q, \eta_{t+1}) - MAP(q, \eta_t)$. With this reward function the agent has to learn to maximize the change in the IR model's performance, $MAP(q, \eta_T) - MAP(q, \eta_0)$. Consider the following,

$$\begin{aligned} R_{t+1} + R_t &= MAP(q, \eta_{t+2}) - MAP(q, \eta_{t+1}) && (1)\\ &\quad + MAP(q, \eta_{t+1}) - MAP(q, \eta_t) && (2)\\ &= MAP(q, \eta_{t+2}) - MAP(q, \eta_t), && (3) \end{aligned}$$

for $t \in [0, T-1]$. Therefore, when $\gamma = 1$ the agent maximizes the return,

$$\begin{aligned} \sum_{t=0}^{T-1} \gamma^t R_t &= \sum_{t=0}^{T-1} \gamma^t (MAP(q, \eta_{t+1}) - MAP(q, \eta_t) && (4)\\ &= \sum_{t=0}^{T-3} \gamma^t (MAP(q, \eta_{t+1})) - MAP(q, \eta_t) && (5)\\ &\quad + MAP(q, \eta_T) - MAP(q, \eta_{T-2}) && (6)\\ &= MAP(q, \eta_T) - MAP(q, \eta_0). && (7) \end{aligned}$$

This property can be made to hold true for all $\gamma \in [0, 1]$ if the reward function is modified to be $R_t = \gamma MAP(q, \eta_{t+1}) - MAP(q, \eta_t)$. This reward function is a potential based reward function similar to that used in reward shaping [22][1]. However, when $\gamma < 1$ the agent is tasked with maximizing $\gamma^T MAP(q, \eta_T) - MAP(q, \eta_0)$, which does not reflect our intended objective. In our evaluation of agent performance using R_{diff} we set $\gamma = 1$.

3.1.4 Gamma. We select $\gamma = 0.995$ for L4 and $\gamma = 0.999$ for Robust04. While $\gamma = 1$ follows from the reward shaping formulation, its role in the MDP heavily influences the difficulty of training the agent. The selection of γ reflects a balance between ease of learning a policy with a smaller γ and the performance of the policy using a larger γ.

4 AGENT

In this section, we discuss the agent and the approach used to learn a policy.

Algorithm 1: Approach for learning a policy based control method.

Input: Episode limit L, IR Model f_{IR}, Training Data, D_{tr}, and Validation Data ,D_{tr}^{val}, Reward function R, Stop Condition *stop*, and initial policy π

for *episode l = 1 to L* **do**
- t = 0
- initialize state s_0 from f_{IR}, D_{tr_t}
- **while *not*** $stop(f_{IR}, D_{tr}^{val})$ **do**
  - $a_t \sim \pi(s_t)$
  - Take action a_t and generate negative set D^-
  - Update f_{IR} with $\{D_{tr_t}, D^-\}$
  - Observe $s' = s_{t+1}, r \leftarrow R(f_{IR}, D_{tr}^{val})$
  - Update Agent via Eq. 9,10,11
  - $t \leftarrow t + 1$

[1]To make R_{diff} a potential based reward that preserves optimal when added to another reward function the following reward function can be used: $\gamma\varphi(\eta_{t+1}) - \varphi(\eta_t)$, where $\varphi(\eta_k) = \gamma MAP(q, \eta_k) - MAP(q, \eta_0)$

4.1 Agent Architecture

Given the distributed representation of $\mathcal{S}$, neural models are prime candidates for this approach due to their ability to transform the input space into a linearly separable decision boundary. However, the depth of the network is a critical component as non linear function approximations significantly contribute to divergence when learning a policy. Thus, we choose a shallow network to reduce stability issues shown in Figure 1[2]

As each state is generated from a standard minibatch with only relative locations playing an important role, we adopt a convolutional perspective. This approach suits the $\mathcal{S}$ distribution as there are two stages of processing. First, a small two dimensional convolutional kernel is used to match neighboring $\phi(q_i), \phi(d_{i_r})$ embeddings in $\mathbf{B}_i$. This is convolved with a large $\frac{|b|}{2}$ filter size with zero padding to learn a general batch difficulty representation. This representation is then max-pooled and passed to three affine transformations of dimensions [500,$|\mathcal{A}|$, $|\mathcal{A}|$]. As discussed in the following section, a softmax is then taken and sampled to determine the action a_i. This model architecture is used for both the actor and critic with the exception of the final layer, which maps to a scalar value of the state for the critic.

4.2 Policy Gradient

Many RL methods exist to optimize the agent's policy, π. In this work, we focus on the family of algorithms known as policy gradient methods. That is, we employ algorithms which approximately optimize $J(\theta)$ via gradient ascent with respect to policy parameters θ, *i.e.*, $\nabla J(\theta)$. An expression for this gradient is given by the policy gradient theorem [30], which, states that

$$\nabla J(\theta) = \sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in \mathcal{A}} \pi(s, a, \theta) q^{\pi}(s, a) - b(s) \frac{\partial \ln \pi(s, a, \theta)}{\partial \theta}, \quad (8)$$

where $d^{\pi}(s) := \sum_{t=0}^{\infty} \gamma^t \Pr(S_t = s)$ and $b(s)$ is a state dependent baseline, i.e, an estimate of the value function $v^{\pi}(s)$. In practice, one can use the REINFORCE algorithm [35] to estimate the gradient using Monte-Carlo simulation, replacing $q^{\pi}(s, a)$ with the observed return from state s and taking action a. However, this method has high variance and updates after an episode which makes it a poor fit for this setting.

The Actor-Critic algorithm [29] is a policy gradient method that performs online updates and estimates $q^{\pi}(s, a)$ with a lower variance, but biased approximation. An actor-critic algorithm is composed of two parts — an actor (the policy) and a critic that evaluates the quality of actors choices. The critic component in this work uses a neural network to approximate the state value function and is trained with temporal difference learning (TD).The TD-error, δ_t, is the difference of predicted value of the state S_t and the prediction after observing the reward R_t and next state S_{t+1}, i.e., $\delta_t = R_t + \gamma f_v(S_{t+1}) - f_v(S_t)$, where f_v is a function that estimate $v^{\pi}(s)$ with weights $v \in \mathbb{R}^n$. This prediction error is used to update both the policy, π, and function approximator f_v. The following

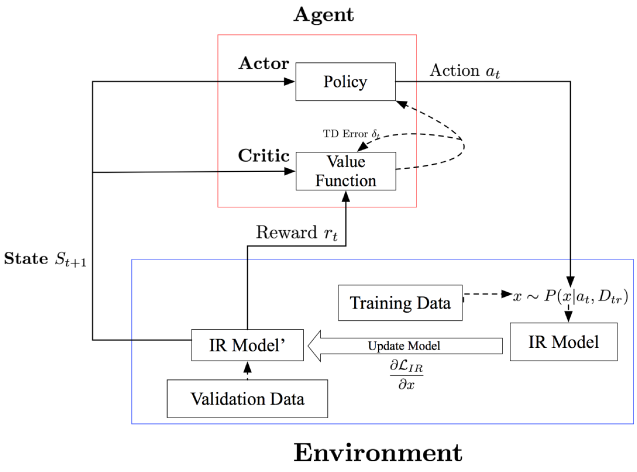

Figure 2: Actor-Critic Setup with IR Environment

updates define the actor critic algorithm:

$$\delta_t = R_t + \gamma f_v(S_{t+1}) - f_v(S_t) \quad (9)$$

$$v = v + \alpha_v \delta_t \frac{\partial}{\partial v} f_v(S_t) \quad (10)$$

$$\theta = \theta + \alpha_\theta \delta_t \frac{\partial}{\partial \theta} \ln \pi(S_t, A_t, \theta) \quad (11)$$

where α_v and α_θ are positive scalar learning rates.

In addition to the actor-critic algorithm we also experiment with the Proximal Policy Optimization (PPO) [28], an off-policy actor critic algorithm. Off-policy algorithms optimize the policy π using data collected from some other policy π_{old}. These algorithms introduce an additional optimization challenge as the distribution of data collected using π_{old} does not match the data distribution of π. This distribution mismatch can be corrected using importance sampling [25], but increases the variance of the gradient estimates. In our experiments below, actor-critic outperforms PPO.

5 EXPERIMENTS

In this section, we describe the baseline approaches, data used, and IR models evaluated. Succinctly, we train and evaluate the learned policy on two different IR models over two different collections. Once the agent has converged as evidenced by performance of the IR model on the validation set, we evaluate the IR model's performance.

5.1 Collections

We use two diverse collections, each representing a different negative sampling choice. The first collection, Yahoo's Webscope L4 [3] represents an answer passage retrieval problem, where there is only one relevant answer in the entire collection. These questions are filtered from Yahoo Answers that meet the criteria of manner questions, as discussed in [31] and has approximately 120,000 query-answer pairs. This collection represents the situation when BM25 is not a strong baseline for relevance [6, 31] and includes the

[2]https://github.com/dscohen/NegSampleAC.git

[3]https://webscope.sandbox.yahoo.com

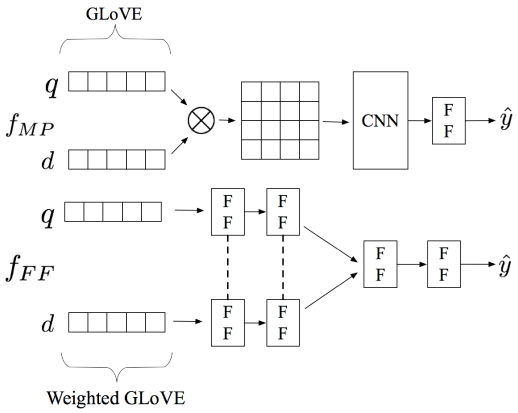

Figure 3: Compressed architecture of two neural models used. FF represents a feedforward layer, f_{MP} is MatchPyramid while f_{FF} is a siamese network with dotted lines representing shared weights. GloVE weighting layer is excluded in diagram for f_{FF}

task of identifying effective negative passages for training as there is only one positive passage with all others acting as negatives.

The second collection, Robust04, consists of 500k news documents from TREC disks 4 and 5, and the query set consists of 250 title queries (TREC topics 301-450, 601-600). Here, each query has judged negative documents along with multiple relevant documents. Furthermore, these judged documents were selected from pooled retrieval runs that are heavily influenced by term frequency information. Thus, this represent the case where labeled information is significantly richer, but there exists an underlying bias as well as much smaller query set for training.

5.2 Baselines

We examine naive sampling over all functions in $\mathcal{A}$. Once each function retrieves a list of candidate negative documents, three approaches are examined. First, a naive random sampling approach is used over the list. Second, we adapt an uncertainty sampling based approach, expected error reduction (EER), to select the best negative document to train on[27]. As EER is performed for active learning where $P(y|D) >> 0$ for some class y, this is not true in search, where the majority of a collection is not relevant to a given query. Thus, given a random document, we assume its correct label is non relevant and arrive at the below representation of EER adapted to a known class:

$$d_{neg} = \arg\max_{d \in C \setminus R_q} P_\eta(r|q,d) \log P_\eta(r|q,d)$$

where $C \setminus R_q$ represents the set of documents not labeled relevant for query q, and $P_\eta(r|q,d)$ represents the probability given parameters η that document d is relevant to query q. As the third sampling baseline, we implement an approach leveraging a distribution based view of information gain [38] referred to as Dynamic-λ.

In addition, as the policy will have access to a larger sampling space than some of the baselines, we include a random agent to ensure that the policy has learned something besides a random sampling approach. This is a relatively high baseline given the curated $\mathcal{A}$ function space as well as a random policy acting as a competitive baseline in [14].

Lastly, we include the performance of IRGAN [32] on only the discriminator. While the authors state that IRGAN consists of two types of retrieval model and often one outperforms the other, it can be viewed as learning the most difficult sampling policy via REINFORCE for the discriminator. Thus, we ignore the generator to properly compare frameworks. We adopt code provided by the authors and tune for performance on each model and collection.

5.3 Neural Retrieval Models

We evaluate the efficacy of sampling methods with two deep neural methods that provide a challenging control problem for the policy due to the large numbers of parameters. We introduce two neural models of varying complexity as seen in Figure 3.

First, we introduce a feedforward model, referred to as f_{FF} that is typical of a distributed neural retrieval model [19]. Using a GloVE initiated embedding, f_{FF} treats the document as a weighted bag of words similar to [7] with a learned weight for each term. This is then averaged into a vector representing each query and candidate document. The lower layers, referred to as f_l, are of dimension [2048,1024,512] and process the query and document independently prior which are then concatenated to form a query-document vector $< f_l(Q), f_l(D) >$. This is then passed into an upper feedforward network of dimension [512,300,1]. f_{FF} is trained by maximizing the log likelihood of the correct document over the sampled set, C_s, via taking the softmax.

As f_{FF} leverages an independent query-document representation, we implement Match Pyramid (MP) using a cosine similarity function over input embeddings to demonstrate an interaction based model under policy control [23]. The model is trained as in the original work by using a cross entropy loss function to learn parameter weights and is referred to as f_{MP}.

5.4 Evaluation

We evaluate the policy by examining performance on the held out validation set during each episode. We examine MAP of the top ranked 100 documents for Webscope L4 and the top 1000 for Robust04. The smaller scope of L4 was selected due to the computation costs as there are roughly 12,000 queries in the test set. Significance between methods are determined via two tailed t-test. However, as we are examining not only total performance, but consistency across different random seeds and hyperparameters, we include the Kolmogorov-Smirnov test that measures the probability of two empirical distribution functions belonging to the same distribution. Both measurements are evaluated with a significance value of $p < 0.05$.

6 RESULTS

In this section, we evaluate the method, AC-IR, over two different retrieval tasks and neural models. We report distribution information by including mean and standard deviation. As discussed in in previous work [3, 16] and bolstered by the lottery ticket hypothesis [10], examining the max run from multiple experiments leads to an ineffective evaluation of a stochastic process. After discussing indicators of performance, we provide analysis into the impact of

the reward shaping approach, hyper-parameter stability, and lastly, the convergence and stability of the agent over multiple runs and during training.

Method	Webscope L4	
	f_{FF}	f_{MP}
BM25$_{rand}$	0.0706±.029	0.1631±.064
BM25$_{Dynamic-\lambda}$	0.0905±.032	**0.2083±.040**
BM25$_{EER}$	0.0919±.031	0.2050±.039
Random$_{rand}$	0.0679±.004	0.0727±.011
Random$_{Dynamic-\lambda}$	0.0621±.073	0.0915±.005
Random$_{EER}$	0.0642±.065	0.0899±.005
IRGAN$_{policy}$	0.0557±.038	0.0807±.005
AC-IR	**0.1239±.011***†	0.1975±.008†
Random AC-IR	0.0603±.003	0.1026±.040
	Robust04	
	f_{FF}	f_{MP}
BM25$_{rand}$	0.0320±.012	0.056±.015
BM25$_{Dynamic-\lambda}$	0.0408±.014	0.0549±.012
BM25$_{EER}$	0.0409±.012	0.0558±.011
Random$_{rand}$	0.0390±.001	0.0518±.015
Random$_{Dynamic-\lambda}$	0.0401±.002	0.0597±.022
Random$_{EER}$	0.0642±.065	0.0600±.021
IRGAN$_{policy}$	0.0394±.006	0.0538±.019
AC-IR	**0.0496±007***†	**0.107±.046**
Random AC-IR	0.0455±.003	0.102±0.048

Table 1: Performance of sampling methods with respect to mean average precision. Mean performance is included with standard deviation. *,† refer to significance to $p < 0.05$ compared to highest baseline using Student's t-test and the Kolmogorov-Smirnov test respectively.

6.1 IR Impact

Looking at the runs in Table 1, we observe that AC-IR significantly improves the consistency of performance on Webscope L4 for both f_{FF} and f_{MP} over multiple random seeds. In the case of f_{FF}, AC-IR is able to outperform that of the EER and Dynamic-λ approaches on both collections without explicit access to the model's uncertainty on a new batch. The agent is able to capture this internally using only $\mathcal{S}$ and the reward to infer this information. As an example, we plot performance over many random seeds in Figure 4 on L4, and only AC-IR is capable of consistently achieving the upper bound of performance when compared to other methods. Furthermore, we identify a bimodal distribution on the BM25 baselines, where the model either successfully converges to values near the max for the given mode or fails to learn based off of the initial parameter distribution.

For f_{MP}, AC-IR performs slightly worse than Dynamic-λ on L4 with respect to both mean and distribution characteristics. However, on the case of Robust04, the policy significantly outperforms all baselines, and reaches parity with the random agent. This behaviour is learned, as non-linear RL has a tendency to collapse to a single action [30], and AC-IR is significantly different than the random policy on all other collections. Furthermore, the BM25 methods have a greater increase in reward during initial minibatches, and without an effective policy to converge on, AC-IR would most likely collapse to BM25.

The result of AC-IR not drastically improving the max reported score is particularly interesting, as unlike standard supervised training collections like CIFAR [17], the information space over IR collections is significantly larger with respect to Shannon entropy. This suggests that the neural models are possibly limited by the number of linear regions the parameters can operate over as discussed by Montufar et al. [21]. Thus, viewing the functions in $\mathcal{A}$ as a set of linear regions, the neural IR models are exposed to a well defined but narrow area of the manifold via BM25, and a much larger area via the random process with the possibility that the gradient descent update might not be informative due to multiple linear regions within a minibatch. Therefore the upper bound of each neural model is not significantly improved by AC-IR. However, controlling the type of regions exposed to the model during training significantly improves mean performance, as well as the number of runs that fall near the upper bound of performance.

Lastly, the relatively low performance of IRGAN can be attributed to three issues. First, REINFORCE is high variance given the static state value $b(s)$ in Equation 9 and the fact that the reward can suffer large changes in certain states, such as if the IR neural model begins overfitting on the training set. This is further exacerbated by the depth of the neural retrieval models being used in this experiment. Second, the authors state that the generator applies a hierarchical softmax, but this is a non trivial structuring of the sample space [12]. Third, we do not use the generator as a ranking model as it represents the sampling policy to train the discriminator.

Succinctly, the learned functional policy of AC-IR is able to take advantage of the strong performance of the static policies while ensure the poor performance regions of their distributions are not reached.

6.2 Document Level Actions

While the AC-IR acts over sampling functions, we investigate the capability of the AC agent to learn a policy to select individual documents. We convert all documents in the collection to a $tf.idf$ weighted mean embedding, and for each Q, D in a batch, we create a candidate list of size s_d from the $tf.idf$ weighted embedding of Q via cosine similarity. We use a new action space, $\mathcal{A}_{\mathcal{D}} = \mathbb{N}_{<s_d}$, where the i^{th} action represents selecting the i^{th} closest document in cosine space. We modify $\mathcal{S}$ to include additional information about the candidate list for each query by a matrix $\mathbf{M}in\mathbb{R}^{s_d \times 5}$, where each tf, document length, unique terms, cosine similarity, and BM25 ranking. As over 91% of queries have a top 100 ranked BM25 document within the top s_d documents in cosine space, the agent should be able to at least collapse to a BM25 sampling policy which we empirically determined to be a more effective policy to cosine similarity. However, the agent fails to converge on this new MDP despite extensive hyperparameter tuning. We investigate this behaviour further by incorporating imitation learning to identify what kind of signal is required to learn a BM25 sampling policy in cosine space. Following work by Bojarski et al. [4], we pretrain the AC agent using a supervised signal rather than directly with a reward

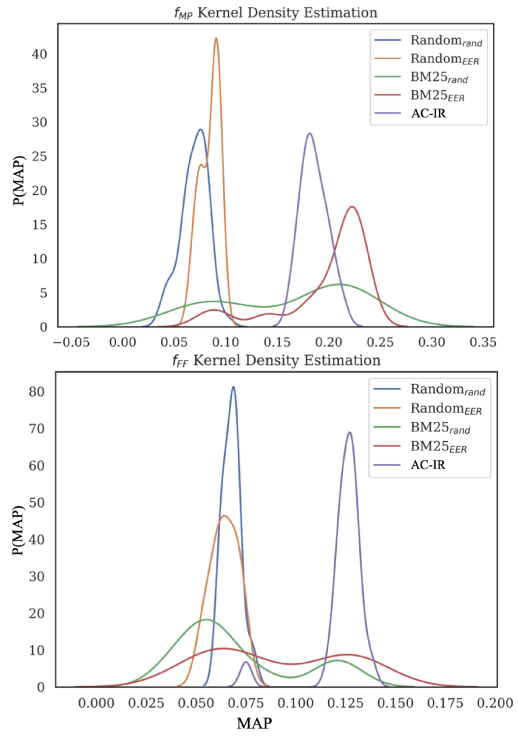

Figure 4: Distribution of policy performance using kernel density estimation over Webscope L4. AC-IR demonstrates performance during convergence.

function. In our case, the signal consists of a binary label for each document indicating whether the document is also within the top n retrieved BM25 ranked documents for the query. In theory, the new state space includes enough information to determine rough BM25 rankings indicated by past work in weak supervision [7]. However, even with the benefit imitation learning the agent still fails to perform better than random.

Examining the generalization properties of the model trained via imitation learning, we observe that the agent is only capable of memorizing the rankings BM25 documents until its parameters are saturated. This suggests that for the case of document selection in an MDP, actions, and thus the corresponding updates to the policy, should be ranked rather than treated in isolation as potential future work.

6.3 Agent Training

Convergence and Stability: Previous work has discussed the instability of both RL and neural networks using the same hyper-parameters over training data [16, 30]. In our case, we observe instability during training of both the neural IR model and the agent. As shown in Figure 4, simply changing the random seed significantly impacts the performance of a neural model on the majority of sampling methods. This presents a challenging problem for the agent as it must not only identify how well the neural model is converging based on the agent's knowledge from past episodes,

Reward Method	MAP	Convergence Rate
$\mathcal{R}_{diff}$	0.1239	31%
$\mathcal{R}_{raw}$	0.0603	0%
$\mathcal{R}_{ceil}$	0.0671	12%

Table 2: Performance of agent trained to optimize different reward functions. MAP on Webscope L4 with convergence rates greater than random are reported using f_{FF}.

but determine the position of this neural model within its performance distribution. The way we defined the state space cannot fully capture the underlying mechanisms of the neural IR models. That is the state of the IR model is only visible to the agent and can be modeled as a partially observable Markov decision process, using the observation function $O : \mathcal{S} \times \mathcal{O} \rightarrow \mathbb{R}_{\geq 0}$, describes the distributions over observations (features), ψ, given a state S. The state as we have formulated it are only observations and the actual state remains unknown to the agent. Currently, function approximation is used to overcome the partial observability and learn a sampling strategy without knowing exactly on which queries and documents the IR model can rank correctly. If instead a set of features that provide this missing information were used, the agent could then learn a policy to that adapts the sampling strategy to explicitly exploit the current state of the IR model.

6.4 Hyper-parameter Case Study

Reward Shaping: We report AC-IR under $\mathcal{R}_{raw}, \mathcal{R}_{ceil}, \mathcal{R}_{diff}$ over the L4 dataset shown in Table 2. The performance of the policy with respect to the IR task of training an effective model is clearly shown to be captured most by $\mathcal{R}_{diff}$, while $\mathcal{R}_{ceil}$ is able to achieve an effective policy for this task albeit in less than 12% of all runs for only a small number of episodes. However, the optimal policy for $\mathcal{R}_{raw}$ is drastically different than what we expected and was similar to findings in [22]. In this case, as the future discounted return $\mathcal{R}_{raw}$ is monotonic with time due to the inability for MAP to be negative, the agent learns to prolong training as long as possible. Rather than maximizing the performance, it avoids triggering the early stopping condition by ensuring small but consistent gains in performance. Given a long enough training period, this policy produces a greater return than attempting to maximize the performance of the neural model at a single epoch. While modifying γ can reduce the total return and make this process easier to learn for the agent, it becomes another critical hyperparameter that is tied to collection and neural model characteristics rather than the overall control problem.

Limiting the Amount of Epochs: As each episode is defined by training the neural model until the stopping condition is met, we experiment with setting a limit on the amount of epochs that can occur in each episode to facilitate faster training of the agent. We set the maximum number of epochs to four, and identify that this results in better performance for the agent learning from $\mathcal{R}_{ceil}$, but prevents an effective policy to be learned under $\mathcal{R}_{diff}$. This result supports the reward shaping motivation discussed in Section 3.1.3, as the hard limit on four epochs relieves the agent operating under $\mathcal{R}_{ceil}$ from stopping the training as early as possible and instead focuses on maximizing performance during this time.

Early Stopping: As the IR model will eventually overfit on the test data regardless of the sampling method, the reward on the held out validation set will start decreasing. This undesirable tail end behaviour can then result in non optimal updates to the policy with respect to the performance of the neural IR model prior to overfitting. Furthermore, the choice of $\gamma < 1$ biases the expected return by discounting the earlier rewards achieved when the IR model was able to generalize to data outside of the training set. We experiment by increasing the early stopping criteria to a patience of ten epochs that fail to improve over the validation set. In doing so, we observe that no agent is able to successfully converge to an effective sampling policy that's significantly different than random. This behaviour suggests that the agent is not capturing the environment fully, as the critic should be able to learn $b(s)$ as shown in Equation 9.

7 CONCLUSION

Motivated by past work in curriculum learning, we propose a novel method to better train a neural IR model over a set of predefined sampling functions. The policy learned by the AC agent is able to mimic the behaviour of uncertainty sampling without the need to process the batch twice over, and results in a better distribution of performance that outperforms established error reductions methods.

8 ACKNOWLEDGEMENTS

This work was supported in part by the Center for Intelligent Information Retrieval and in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA) via AFRL contact #FA8650-17-C-9116 under subcontract #94671240 from the University of Southern California. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

REFERENCES

[1] Qingyao Ai, Keping Bi, Jiafeng Guo, and W. Bruce Croft. 2018. Learning a Deep Listwise Context Model for Ranking Refinement. In *SIGIR 2018*.
[2] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. 2017. Neural Optimizer Search with Reinforcement Learning. In *Proceedings of ICML 2017 (Proceedings of Machine Learning Research)*. PMLR.
[3] Mauro Birattari and Marco Dorigo. 2007. How to assess and report the performance of a stochastic algorithm on a benchmark problem: mean or best result on a number of runs? *Optimization letters* 1, 3 (2007), 309–311.
[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. 2016. End to End Learning for Self-Driving Cars. *CoRR* abs/1604.07316 (2016). arXiv:1604.07316
[5] Long Chen, Fajie Yuan, Joemon M. Jose, and Weinan Zhang. 2018. Improving Negative Sampling for Word Representation Using Self-embedded Features. In *Proceedings of ACM International Conference on WSDM*. ACM.
[6] Daniel Cohen and W. Bruce Croft. [n. d.]. End to End Long Short Term Memory Networks for Non-Factoid Question Answering. In *ICTIR '16*.
[7] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. 2017. Neural Ranking Models with Weak Supervision. In *SIGIR*. ACM.
[8] Yixing Fan, Jiafeng Guo, Yanyan Lan, Jun Xu, Chengxiang Zhai, and Xueqi Cheng. 2018. Modeling Diverse Relevance Patterns in Ad-hoc Retrieval. In *SIGIR*. 375–384.
[9] Yang Fan, Fei Tian, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. 2018. Learning to Teach. In *ICLR*.
[10] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. 2019. The Lottery Ticket Hypothesis at Scale. *CoRR* abs/1903.01611 (2019). arXiv:1903.01611 http://arxiv.org/abs/1903.01611
[11] Yoav Goldberg and Omer Levy. 2014. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *CoRR* abs/1402.3722 (2014).
[12] Joshua Goodman. 2001. Classes for Fast Maximum Entropy Training. In *ICASSP*.
[13] Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2016. Efficient softmax approximation for GPUs. *CoRR* abs/1609.04309 (2016).
[14] Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. 2017. Automated Curriculum Learning for Neural Networks. In *ICML (Proceedings of Machine Learning Research)*. PMLR.
[15] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007* (2014).
[16] Scott Jordan, Daniel Cohen, and Philip Thomas. 2018. Using Cumulative Distribution Based Performance Analysis to Benchmark Models. In *NeurIPS*.
[17] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.
[18] Bhaskar Mitra and Nick Craswell. 2018. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval (to appear)* (2018).
[19] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to match using local and distributed representations of text for web search. In *WWW 17*. 1291–1299.
[20] Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in NIPS*. 2265–2273.
[21] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. 2014. On the Number of Linear Regions of Deep Neural Networks. In *Advances in NIPS*. Curran Associates, Inc.
[22] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of ICML*. Morgan Kaufmann.
[23] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2016. Text Matching As Image Recognition. In *Proceedings of AAAI*. AAAI Press.
[24] Jay M. Ponte and W. Bruce Croft. 1998. A Language Modeling Approach to Information Retrieval. In *SIGIR (SIGIR '98)*. ACM, New York, NY, USA, 275–281.
[25] Doina Precup, Richard S. Sutton, and Satinder P. Singh. 2000. Eligibility Traces for Off-Policy Policy Evaluation. In *Proceedings of ICML*. Morgan Kaufmann.
[26] Chen Qu, Feng Ji, Minghui Qiu, Liu Yang, Zhiyu Min, Haiqing Chen, Jun Huang, and W Bruce Croft. 2018. Learning to Selectively Transfer: Reinforced Transfer Learning for Deep Text Matching. *arXiv preprint arXiv:1812.11561* (2018).
[27] Nicholas Roy and Andrew Mccallum. 2001. Toward optimal active learning through monte carlo estimation of error reduction. In *ICML*.
[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017).
[29] Richard Sutton and Andrew Barto. 2016. *Reinforcement Learning*. MIT.
[30] Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. 2000. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in NIPS*. MIT Press.
[31] Di Wang and Eric Nyberg. 2015. A Long Short-Term Memory Model for Answer Sentence Selection in Question Answering. In *ACL-IJCNLP, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*. ACL, 707–712.
[32] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. IRGAN: A Minimax Game for Unifying Generative and Discriminative Information Retrieval Models. In *SIGIR*. ACM, 515–524.
[33] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. 2017. Learning to Model the Tail. In *Advances in NIPS*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 7029–7039.
[34] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. 2017. Learning to Model the Tail. In *Advances in NIPS*. Curran Associates, Inc., 7029–7039.
[35] Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* 8, 3-4 (May 1992), 229–256.
[36] Jiawei Wu, Lei Li, and William Yang Wang. 2018. Reinforced Co-Training. In *NAACL-HLT*, Marilyn A. Walker, Heng Ji, and Amanda Stent (Eds.). ACL, 1252–1262.
[37] Jiawei Wu, Lei Li, and William Yang Wang. 2018. Reinforced Co-Training. *CoRR* abs/1804.06035 (2018). arXiv:1804.06035
[38] Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. 2013. Optimizing top-n collaborative filtering via dynamic negative item sampling. In *SIGIR*. ACM, 785–788.
[39] Barret Zoph and Quoc V. Le. 2016. Neural Architecture Search with Reinforcement Learning. *CoRR* abs/1611.01578 (2016). arXiv:1611.01578