Master of Informatics

Technische Universität München

Interdisciplinary Project

Reinforcement Learning for Active Learning

Ming Yip Cheung

Master of Informatics

Technische Universität München

Interdisciplinary Project

Reinforcement Learning for Active Learning

Author:	Ming Yip Cheung
Supervisor:	Prof. Dr. Daniel Cremers
1st Advisor:	M.Sc. John Chiotellis
2nd Advisor:	PD Dr. habil. Rudolph Triebel
Submission Date:	October 17th, 2019

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

October 17th, 2019 Ming Yip Cheung

Abstract

There are many state-of-the-art research and applications in Active Learning and Reinforcement Learning. It is, however, actually only a few research to connect these two fields. In this project, we would like to apply reinforcement learning to perform active learning. We will propose and implement a reinforcement learning algorithm that learns to select the desired subset of the MNIST dataset [11] for deep classification learning such that the classification network can achieve a better accuracy at the final state. The Program is implemented in Python with Tensorflow [1].

Contents

1 Introduction

In this project, we combine reinforcement learning with active learning for classification learning purposes. The objective of this project is to experiment with the possibility and the challenges of using reinforcement learning for active learning. We train a reinforcement learning agent to learn to select a small set of training samples d from a bigger set of training samples D. We hope to let the reinforcement learning agent achieving or even outperforming the existing active learning methods, similar to the astonishing results that reinforcement learning has succeeded in other computer science realms.

We mainly focus on stream-based active learning in this project. This means the active learning agent receives and selects samples in a small batch each time instead of in a pool-based setting in which an agent selects samples at once with the whole data sets.

1.1 What is Active Learning?

Active Learning is one special Machine Learning algorithms that can sample data from a large data-set and form a smaller data-set. Active Learning can boost up the training process, raise the training accuracy, or reduce the labeling cost of the data-set.

Active Learning is a sub-field in machine learning. During training, an active learning algorithm interactively queries from the dataset and chooses a subset of the dataset. The reason behind using Active Learning is to overcome the dataset labeling bottleneck. Nowadays, despite getting data is relatively easy, labeling them with human power is rather expensive.

In many cases, it is inefficient, if not impossible, to label every single data and images. It raises the question of finding an efficient algorithm to query a dataset. As the new dataset is only a small portion of the original dataset, it also significantly reduces the time needed in training or executing the algorithms.

1.2 Why do we need Active Learning?

There are mainly some reasons to apply Active Learning in classification learning. Firstly, the selected dataset is only a small subset of the original dataset. The training time of the network can be largely reduced. Moreover, active learning can help to achieve higher

accuracy comparing to a passive learning method. Figure 1.1 Since the active learning algorithm would select the most important samples which lie on the decision boundary and also exclude the unimportant data [23]. The sampling and learning process becomes a lot more efficient.

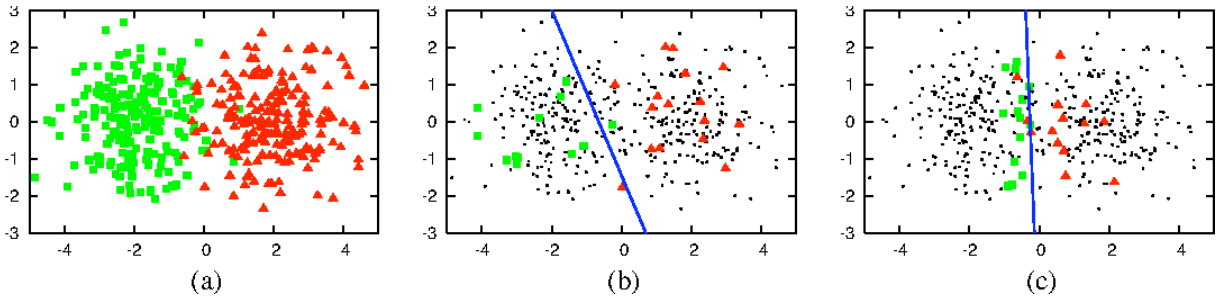

Figure 1.1: An illustrative example of active learning (image taken from [2])

1.3 Reinforcement Learning

Reinforcement Learning is a machine learning algorithm. The algorithm aims to learn the optimal policy such that it can achieve the best-accumulated score in the designated task. We formulate our environment as a Markov decision process. The Reinforcement Learning agent in each move interacts with the environment and receives a reward from the environment, which indicates how good the last action is. The agent learns the best policy from iterative exploring the state-action-reward pairs.

1.3.1 Markov Decision Process

A general assumption behind the reinforcement learning environment is that the transition of state can be formulated as a Markov Decision Process (MDP) [2] [15] . A standard reinforcement learning model usually consists of 5 tuples.

- A set of environment states $\mathcal{S}$
- A set of agent actions $\mathcal{A}$
- A probabilistic transition function of states and actions $\tau(s'|s) = P_r(s_{t+1} = s'|s_t = s, a_t = a)$
- A reward function $R(s, a, s')$ indicates the immediate reward after the transition from state s to s' with the selected action a
- A discount factor $\gamma \in [0, 1]$

At each time t, an agent receives the current environment state s_t and the last transition reward r_t from the environment. If the agent can only partially observe the environment state with observation o_t, we formulate the problem as a Partially observable Markov decision process (POMDP) [24].

An agent in each step chooses an available action a_t according to its policy function π and interacts with the environment. The policy function can be formulated as a deterministic function $\pi(s) \in \mathcal{A}(s)$ and a stochastic function such that the agent picks each action with probability $\pi(\mathcal{A}_t = a|\mathcal{S}_t = s)$. The environment then transitions to a new state s_{t+1} based on the current state s_t and selected action a_t. A new reward r_{t+1} and a new state observation are fed to the agent, so a new cycle starts.

The goal of the reinforcement learning agent is to find an optimal policy function π^*, which maximizes the expected return rewards of all roll-out states of a game:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[G|\pi] \tag{1.1}$$

where $G = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ is the return at time t.

1.4 Reinforcement Learning algorithms

Reinforcement Learning algorithms can be roughly sorted into two big categories: value-based methods and policy-based methods.

1.4.1 Value-based methods

Value-based methods aims to learn an approximate of the action-value function $Q(s, a; \theta)$. One of the most well-known value-based methods is the Q-learning method [26]. The deep learning version DQN [16] is invented in 2015. Value-based method updates its parameters by using and bellman equation and minimising the mean-squared loss.

$$L = \mathbb{E}[(R_t + \gamma max'_a Q(s', a') - Q(s, a))^2] \tag{1.2}$$

1.4.2 Policy-based Methods

In value-based methods, actions are selected based on the action-value estimates. It thereby implicitly defines policies for a game. In contrast to value-based methods, policy-based methods aim to search for and learn the optimal policy π^* directly. The policy is formulated as

$$\pi(a|s, \theta) = Pr\{A_t = a|S_t = s, \theta_t = \theta\}. \tag{1.3}$$

The policy π represents the probability of selecting an action a at state s with the policy parameter θ. We call a policy to be an optimal policy π^*, when the parameterized policy

maximizes the expected return of a game $\mathbb{E}[R|\theta]$.

2 Related Work

2.1 Active Learning

There are many different types of learning extensions aiming to reduce the number of labeled samples. Moreover, active learning methods are suitable for serving as benchmark algorithms. We can use them to compare with our reinforcement learning agent, such as uncertainty sampling [13] [25], least confident method [12], Query-By-Committee [21], variance reduction [18], and etc. In this project, we take the least margin sampling method [19] as a benchmark to compare with our result.

2.2 Deep Active Learning

Deep reinforcement learning (DRL) methods like deep-Q learning [16] combined with experience replay [14] have become a hot topic and has been applied to everywhere. However, not so much researches have connected reinforcement learning with active learning. Recently there are some newly attempts to use deep active learning to use in transfer learning [7], in named entity recognition [22] or in data augmentation [5].

2.3 Time Limits In Reinforcement Learning

Our approach is also inspired by an idea from Time Limits in Reinforcement Learning [17]. [17] states that terminations due to budget limits are part of the state of the environment. Thereby, in order to avoid violating the Markov property, we should include the remaining time (budget) as part of the state.

They showed that in their experiment, time-awareness could improve performance dramatically. Agents learn effectively to maximizing the travel distance to achieve a higher score before the end of the game.

2.4 Partially Observable Markov Decision Process

Reinforcement Learning based on the Markov Decision Process is a powerful tool. However, in many more cases, agents can only observe part of the environment. And hence, they suffer from limited observation. Partially Observable Markov Decision Process [24]

provides a new framework to deal with the uncertain of the state.

In our project, our agents also suffer from limited observation ability. Although when we implemented our algorithm, we still formulated as a standard Markov Decision Process, it is still worth to mention that there is a potential improvement to raise the performance of the agents.

3 Methodology

3.1 Training scheme

To train a reinforcement learning agent to work as an active learning agent, we first need to define the five very important tuples for the problem. In a standard model for reinforcement learning, an RL agent takes actions in an environment. After each transition step, the environment returns a reward and a new state observation. The classification network f_θ serves as part of the environment model.

In the initialization phase, We first split the dataset into three sets - training set, validation set, and test set. We also initialize the reinforcement learning network weights ϕ, and the classification network weights θ and store them for later use.

In the training phase, the RL agent F_ϕ and the environment work closely together and pass information to the other counterpart to complete one training step. In each training step, our classification network θ provides the state S_t and reward R_t information for the reinforcement agent. The reinforcement agent reacts with the environment by choosing an actions A_t from a set of available actions.

3.1.1 A complete agent training step

Now, let us have a closer look at a complete agent training step, as shown in Figure 3.1. Firstly, the environment agent randomly picks K images from the dataset. In the second step, we feed this K images batch into the classification network and get the value of the output layer after the softmax function. We also call this value to be the output probabilities y of the classification network.

In the third step, we feed the output probabilities y to the RL network F_ϕ. The RL network then predicts the accumulated future return G_t for each image. We then select the top-k images with the highest predicted future return. The rest of the samples are discarded.

Fourthly, the classification Network f_θ trains with the selected images k for some epochs. We then get a validation accuracy A_t of the classification network. In step 5, the environment agent sends the accuracy A_t as a reward together with the new states S_{t+1} (new

output probabilities of images) to the RL agent. Lastly, the RL agent uses the state-reward pairs to train and update its network.

$$V(S) \leftarrow V(S) + \alpha(\underbrace{r + \gamma V(S')}_{\textit{The TD target}} - V(S)) \tag{3.1}$$

After that, a training step is completed, and a new cycle begins. When the classification network has trained with as many images as the budget B, a game episode is completed. We reset the environment counters. Moreover, the classification weights θ are also reinitialized with the weights we stored before. A new game episode begins, and we continue to train the RL agent with the newly reinitialized environment. When the RL agent weights have converged, the training phase is completed.

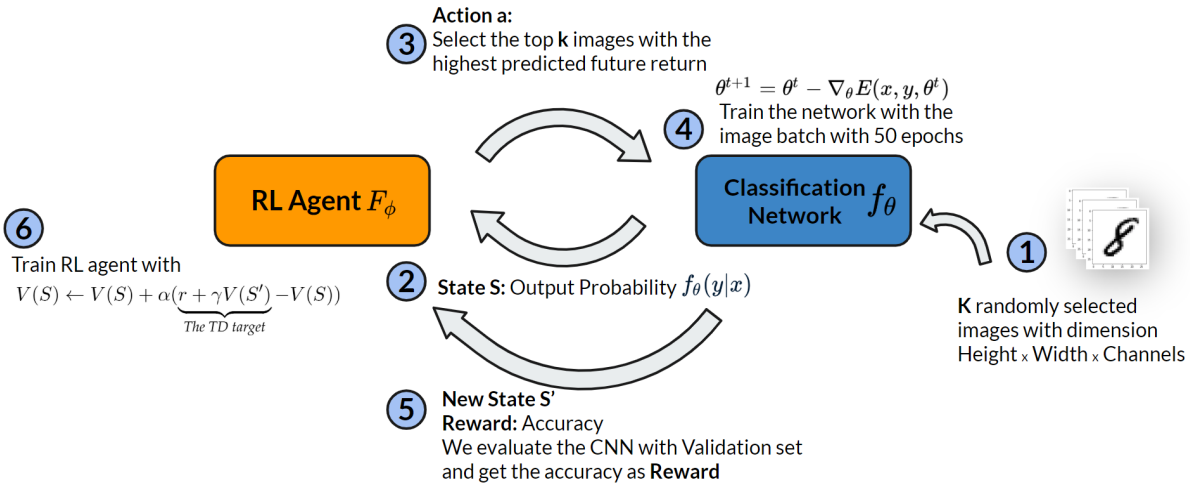

Figure 3.1: One training cycle

3.2 Challenges

3.2.1 Split a complete state into a batch

With the setting mentioned above, we found out several challenges in modeling reinforcement learning for active learning. First, the dimension of a state S is too large. It makes the input dimension of the fully-connected RL network large, and training becomes hardly possible. The input dimension of a RL network for a K samples batch is $W_{(width)} \times H_{(height)} \times C_{(channels)} \times K_{(samples)}$.

Instead of combining every output probabilities state as a whole state s_t, we split the output probabilities into a batch. Now, the input dimension for the RL agent is equal

to the dimension of one output probability value, which is the number of classes in the dataset. We reduce the dimension from *number of class* $\times$ *k samples* to only *number of class*.

One biggest drawback of splitting the state into batch is that the RL agent gets less information per each run. It only partially observes $\frac{1}{k}$ of the state. In some sense, this setting may transform the original problem to be a more difficult problem. Besides, after the classification training step, all the top k images in the same batch share the same validation accuracy $\mathcal{A}$ of classification network and use this as the reward R to update the RL agent. It becomes inefficient to train the RL network.

3.2.2 Markov Decision Process assumption

Second, a general assumption of a reinforcement learning environment is that the transition of state can be formulated as a Markov Decision Process (MDP), which is an extension of Markov Chain. The transitional probability of an event in a Markov Chain model depends only on its previous attached states.

$$P(S_{t+1} = s_{t+1}|s_t, s_{t-1}, s_{t-2}, ..., s_1) = P(S_{t+1} = s_{t+1}|s_t) \tag{3.2}$$

A naive approach does not necessary hold the Markov Chain property assumption. As the environment agent only randomly selects images from the date-set, each state does not necessary depend on the previous state.

3.2.3 Incomplete information of the state

Third, the RL agent observes only part of the state. Instead of observing only the currently chosen samples, the RL agent should also get information about the weights of the classification network. However, the dimension of a state is massive if the weights of the classification network are attached in the state, making the training process more difficult.

3.2.4 Exponential growth of the action-value space

Fourth, a Q-function approximation $Q(s, a)$ takes in two parameters - state s and action a. The action space for a K samples batch is $A \in \{0, 1\}^K$. The action space grows exponentially. Therefore, it is never possible to learn the Q-function approximation directly.

As we know, the dimensionality of the Q-function $Q(s, a)$ is far much larger than the Value function $V(s)$, it is more efficient to learn the state-value. So we modify and make

the RL agent to get the output probabilities y, which are the classification network output values of the K samples batch selected by the environment agent, as the state s_t. The RL agent takes in the states information and estimates the expected future return $V(s) = \mathbb{E}[G_t|S_t = s]$. We define the RL actions by selecting the top k samples with the highest predicted values $V(s)$. These selected k samples are used to train the classification network.

These modifications significantly reduce the dimensionality of the state space and hence, make the training for RL agent feasible and match the theory of reinforcement learning.

3.3 Budget

Here, we explain more about the details of introducing a budget counter in training. Since the objective of active learning is to select a subset sample of a data set and achieve good training results, it is necessary to set a budget for the reinforcement learning agent. Otherwise, a reinforcement learning agent does not have any limitation and selects every single sample for training. Although selecting all samples may not guarantee to have an optimal training set for the classification network, it is not hard to imagine with the full data set, the classification accuracy after training can be high. An agent, in this case, may have no incentive to improve its policy but only selecting every samples.

Therefore, for all the experiments we implemented, we set a Budget B for the reinforcement agent. A budget counter keeps tracking of the number of selected samples. When the chosen samples exceed the budget B, a game episode ends, and we reinitialize the classification network weights. A new training episode for the reinforcement agent begins.

We also need to pose a Budget limitation on the environment side. The reinforcement agent still has no information about the budget and the remaining selection quota. An idea suggested by [17] provides us a solution for value-based methods to train a time-aware agent. They suggested that the remained time limit should be attached to the state information and be fed into the state-value function. Therefore, to adapt the above modification to our project, for every episode step, we attach the remaining episodes and the remaining classification budgets along with the state information. The new state-value function at episode t and classification step i in an environment with a time limit T and game budget B :

$$V_\pi(s, T-t, B-ss) = \mathbb{E}_\pi[G_{t:T}|S_t = s] \tag{3.3}$$

The target y for a one-step temporal-different(TD) update is now updated to be:

$$y = \begin{cases} r & \textit{at all terminations} \\ r + \gamma \hat{v}_\pi(s', T-t-1, B-ss-1) & \textit{otherwise} \end{cases} \tag{3.4}$$

Note that the current number of selected samples ss = current classification step i × RL agent selection batch size b.

3.4 Algorithm

The full pictures of the reinforcement learning algorithms. 1

Algorithm 1: Reinforcement algorithm for training Value function ϕ

Input : data D, budget B, RL selection batch size b, RL network weights ϕ, classification network weights θ
Output: RL network weights ϕ

Train RL network weight ϕ ;
Fix the samples' order of the data-set D
$steps = \frac{|B|}{|b|}$
for *episode in 1 ... episodes* **do**
  Reset the classification weight θ with same initial random weights
  for *t = 0, 1, ..., steps* **do**
    Predicted Future Expected Return $V_i(x_t) \leftarrow F_\phi(f_\theta(y|x), T-t, B-i*b)$
    Select top b images $D_t = Top_b\{(x_t, y_t)\}_{i=1}^{b}$
    for *epoch in 1 ... epochs* **do**
      Train Classification Network $\theta \leftarrow \theta - \nabla_\theta L(y_i, f\theta(x_i))$
    end
    Get Validation set Accuracy $R_v = A_i$
    Train RL agent $\phi \leftarrow \phi + \alpha \nabla_\phi F_\phi(S_t, R_t, S_{t+1}, \gamma)$
  end
  Get Final Validation set Accuracy $R_t = A_i$
  if R_t *is the best Accuracy* **then**
    Store the best Agent ϕ
  end
end
return best RL Agent ϕ

Algorithm 2: Evaluation for the Best Agent

Input : data D, budget B, RL selection batch size b, RL network weight ϕ, classification network weight θ

Output: the Average Validation Accuracy

Train RL network weight ϕ ;

Restore the best RL Agent weights from memory **for** *iteration in 1...20* **do**

  Reset the classification weight θ

  for *t = 0, 1, ...,* $steps$ **do**

    Select top b images $D_t = Top_b\{(x_t, y_t)\}_{i=1}^{b}$

    Train Classification Network with the selected images

  end

  Get Validation set Accuracy $R_t = A_i$

end

Get the Average Validation Accuracy

4 Experiment

4.1 Implementation

The algorithm is implemented end-to-end in TensorFlow [1] and Keras [3]. The loss is minimized using Adam [9] for the reinforcement learning network and RMSProp for the active learning network [6]. To optimize the program code, we used a python profiler - Tuna [20] to profile our program run time and to optimize the bottleneck part of our code to speed up the training process. We also implemented a ϵ-greedy method for exploration during training and experience replay to more efficiently use samples and to achieve a more identical and independently distributed dataset.

4.2 Experiments

To appraise the performance of a reinforcement learning agent, we need to have some benchmarks for performance evaluation. It is obvious that we should have a random agent who selects samples randomly as a passive learning method. Thus, any meaningful proposed methods should outperform the random agent and achieve a better terminal accuracy than the random agent. On the other hand, we can also compare our reinforcement learning agent with numerous machine learning methods, for example uncertainty sampling [13]. Here, in this project, we picked the least margin sampling method [19], also denoted as Best-vs-Second-best-value method.

Least Margin Sampling Method (BVSB):

$$x^* = \arg\min_x P_\phi(\hat{y_1}|x) - P_\phi(\hat{y_2}|x) \tag{4.1}$$

In order to examine our algorithm, we performed different experiments on various data sets, including MNIST[11], CIFAR10 [10], and EMNIST [4]. We used the same 20 seeds(0, 1, 2, ..., 19) to initialize the Tensorflow[8] and Numpy pseudo-random number generators for the environment and agent. For the classification network, We used different neural network architectures for various classification networks to suit for the difficulties of learning the datasets. For the reinforcement learning network, we used the following network architecture for all data sets.

Reinforcement Network Architecture:

$$I(\textit{number of class} + 2) \longrightarrow FC(128) \longrightarrow FC(32) \longrightarrow FC(16) \longrightarrow FC(8) \longrightarrow O(1)$$

Here, $I(n)$ stands for an input layer with n attributes. Here the input dimension is *number of classes* + 2 extra features, as we also attach the remaining episodes and the remaining classification budgets to the output probability features as the state information. A rectified linear unit (ReLU) is added in between every fully connected layer. $O(1)$ stands for an output layer with one output value (the predicted expected future return R). $FC(n)$ indicates a fully connected layer with n output units.

MNIST

The Modified National Institute of Standards and Technology database (MNIST) data set contains 70,000 handwritten digits. The handwritten digits are normalized and cropped in black and white images with 28x28 pixels. We split the MNIST dataset into three sets - training set, validation set, and test set with images 50,000, 10,000, and 10,000, respectively.

We perform the experiments as for MNIST with the following classification architecture:

$$C(32, 5) \longrightarrow P(2) \longrightarrow C(64, 5) \longrightarrow P(2) \longrightarrow FC(1024) \longrightarrow FC(10)$$

Here, we use $C(n, k)$ to denote a convolutional layer with n kernels of size $k \times k$. $P(k)$ denotes a max-pooling layer with pooling window size $k \times k$ and stride 2. We train the MNIST classification architecture above with the method mentioned from Section 5.8.3 and achieve interestingly good results as shown in *Table* 4.1. We also plot the accuracy of the classification model per step during the training period on the MNIST data-set with budget 1000 samples, batchsize 10 with 4 different methods *Fig:* 4.1.

The **Random agent** achieves 89.74% and the machine learning active learning agent (**BVSB agent**) achieves 94.47% accuracy. It is worth it to mention that we discover the reinforcement agent RL-top, who selects samples in the top-k prediction value set, performs very poor in the training achieving only 49.24% accuracy. This agent from now on, we denote it as **RL-top agent**.

We also have another discovery. We run another experiment to train a reinforcement learning agent RL-low, who selects samples in the low-k prediction value set. We call this agent in the following section as **RL-low** or **RL agent**. The RL-low agent achieves an unexpected excellent result comparing to the result of the BVSB agent with 94.87%. The above result contradicts our reinforcement learning theory. According to the theory, the

RL-top agent should achieve better accuracy than the RL-low agent as we pick the images with higher prediction values. Those images should be more important in training and contribute more to the learning process. In Section 5.8.3, we explore and try to investigate the cause of the result.

Table 4.1: Average terminal accuracy of 20 experiments on 3 data-sets

Dataset	Random	BVSB	RL-top	RL-low
MNIST[11]	0.8974 ± 0.0240	0.9447 ± 0.0128	0.4924 ± 0.0763	*0.9487 ± 0.00975
CIFAR[4]	0.2238 ± 0.0173	0.2294 ± 0.0253		*0.245714 ± 0.0161
EMNIST[10]	0.7574 ± 0.0104	*0.7975 ± 0.0123	0.7282 ± 0.0232	0.7748 ± 0.0150

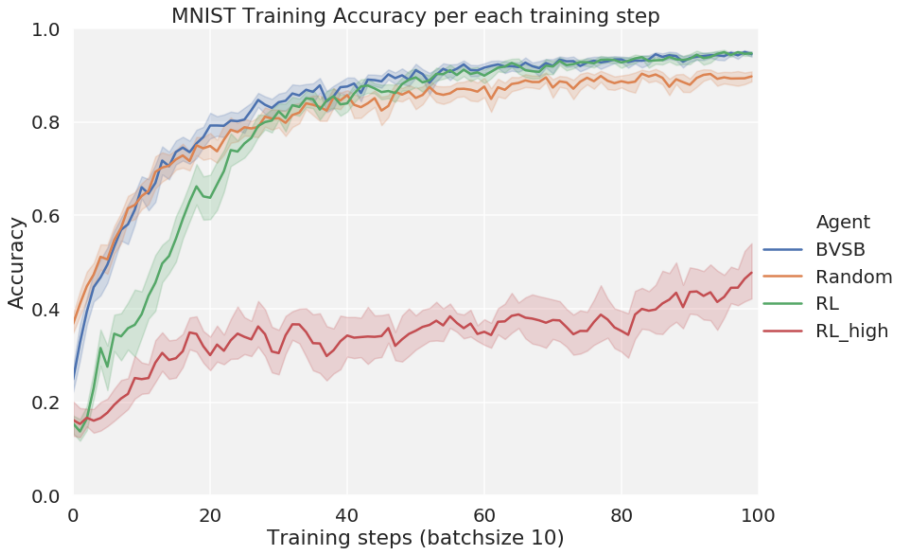

Figure 4.1: Accuracy of the classification model during training on the MNIST data-set with budget 1000 samples, batchsize 10 in 4 different methods.

EMNIST

EMNIST dataset is an extended dataset of the MNIST dataset. It contains up to 62 different classes of handwritten alphabet-digits images. In this project, we use 26 Capital letter classes in the EMNIST Balanced dataset. Each class contains 3,000 images. In total, we use 78,000 images. Similar to the previous setting, We split the dataset into three sets - training set, validation set, and test set with images 62,400, 7,800, and 7,800, respectively.

We perform the experiments as for EMNIST with the following classification architecture:

$$C(32,3) \longrightarrow C(64,3) \longrightarrow C(128,3) \longrightarrow FC(128) \longrightarrow FC(26)$$

Similar to the findings in the MNIST experiments, we found out that BVSB agent and RL-low agent perform the best with accuracy 79.75% and 77.48% respectively Table 4.1 *Figure:* 4.2. The RL-top agent performs still worse than the random agent.

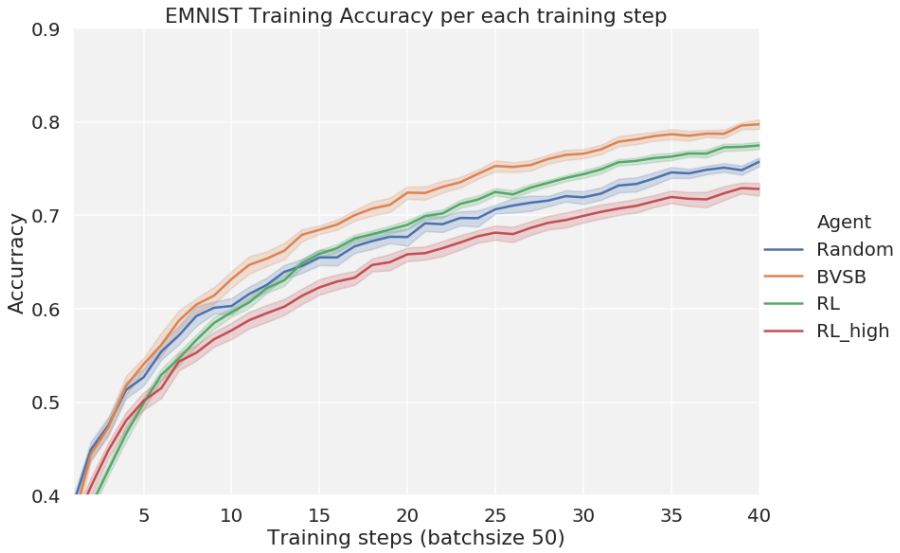

Figure 4.2: Accuracy of the classification model during training on the EMNIST-Balanced data-set with budget 2000 samples, batchsize 50 in 4 different methods.

CIFAR10

We have also tested our algorithm on the CIFAR-10 dataset. It contains 60,000 32x32 RGB images in 10 different classes. We split the MNIST dataset into three sets - training set, validation set, and test set with images 48,000, 6,000, and 6,000, respectively.

We perform the experiments as for CIFAR-10 with the following classification architecture:

$$C(32,3) \longrightarrow P(2) \longrightarrow C(64,3) \longrightarrow C(64,3) \longrightarrow P(2) \longrightarrow FC(512) \longrightarrow FC(10)$$

The random agent, BVSB agent and the RL agent achieve roughly a same accuracy Table

4.1 *Figure:* 4.1. Hence, we found out that it is hard to conclude if the RL agent performs better or worse than the random agent and BVSB agent. We believe that this may be affected by the training difficulty of CIFAR-10.

One reason to explain the result is that the CIFAR-10 does not get any improvement with the BVSB agent. This may imply that there is barely any rooms for the RL agent to learn and achieve better performance.

Moreover, we train the classification network with 50 epochs for each batch selected by an agent with k images. This is, however, we find out that repeatedly learning a same small batch with a large number of epochs(e.g > 50 epochs) may hurt the learning of the classification network. The classification network forgets the previous learning.

Unfortunately, if we reduce the number of training epochs of the classification network, the increase in accuracy of the classification network after each training step becomes insignificant, causing a reward of an action trivial. A reinforcement learning agent in this situation can hardly learn an important state-value function.

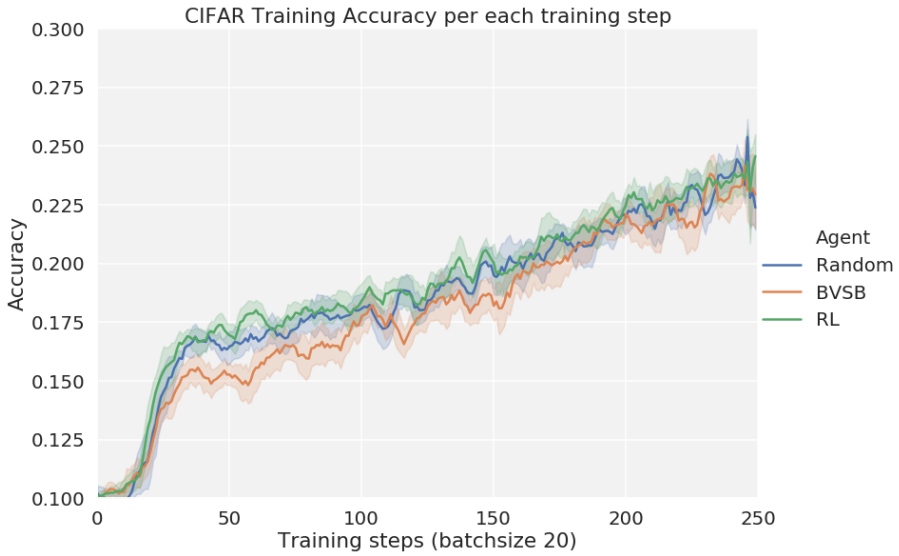

Figure 4.3: Accuracy of the classification model during training on the Cifar-10 data-set with budget 5000 samples, batchsize 20 in 3 different methods.

4.3 Selection Bias Analysis

To explore the performance difference of the RL-top agent and RL-low agent, we plot 12 histogram graphs Figure 4.4 Figure 4.5 of the distribution of 12 different selection batches with budget 1000 on the MNIST dataset. This means after we train both RL-top agent and RL-low agent, we use the agents to train a new initialized classification network under 12 different seeds. We count for each class the frequency of being selected.

At the bottom of each graph, we also stated the final accuracy of classification networks after trained with 1,000 images. We find out that the RL-top agent favors selecting 1 to 2 digits instead of selecting digits in a more even distribution. In our experiments, 8 of 12 runs have a class with more than 50% samples of the total number of samples. All training is heavily suffered from the biased batches with only $40\% - 60\%$ accuracy. Indeed, only in one seed, it has a relatively even distribution in the selection, and the accuracy of this run is 89.6%.

On the other hand, when we have a look at the selection distribution from the RL-low agent, we find out that it tends to select a more even batch. It does a favor to select some digits slightly more, particularly digit 0, 2, and 6. All experiments with RL-low achieve a significant competitive result with accuracy $> 93\%$.

4.4 Entropy of the selection distribution with different value batches

The above experiment inspired us to study the selection distribution entropy of different prediction value batches to reveal the relation of batch entropy and accuracy.

We run our experiments on MNIST dataset. In the previous setting, the sample selection policy of our RL agents is either always selecting the top-k images or the low-k images. In this experiment, we train and test agents with slightly different policies. We denote them policies 1-10. An agent assigned with *policy 1* means that during the selection phase, no matter for the training, evaluation, or the test phase, the agent selects k images with the lowest value. An agent assigned with *policy 2* means that it selects images with the lowest k to $2k$ values. For the other agents with different policies, they select images according to the rank as well.

Therefore, in total, we have 10 different policies selecting 10 different value batches of images. We then collect terminal accuracy and selection distribution of each agent with respect to the rank of their policy. We plot a graph *Figure:* 4.6 to reveal the relation of entropy and accuracy of each batch.

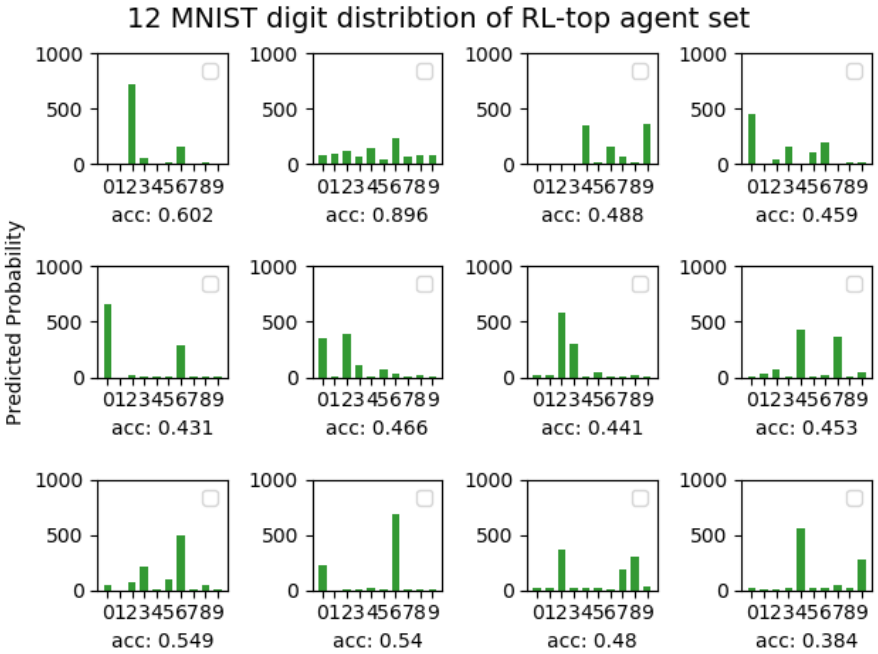

Figure 4.4: Digits Selection Distribution by RL-top agent. *x-axis:* digit class, *y-axis:* the selection frequency of each class

We find out that agents with high-value batch have lower terminal accuracy and entropy in their selection. This means the selection distributions are more biased, and 1 or 2 digits dominates in the sample population. Hence, the classification training may suffer from the biased training set.

4.5 Select one sample from each class method

In order to reduce the selection bias and improve the performance of the RL-top agent, we modify the main algorithm and propose a variation of the algorithm. During the RL agent selection phase, instead of picking the top-K images, the agent picks one image with the highest value from each class.

We implement the new proposed algorithm and test it on the MNIST dataset. We run both RL-top agents and RL-low agents with this modification and rename them as RL-top-balanced agents and RL-low-balanced agents. In *Figure:* 4.7, it shows that both RL-top-balanced agent and RL-low-balanced agent achieve similar scores with 82% accuracy rate. The performance of the RL-top agent does raise to a high level. However, it still underperform the random agent. In contrast, the RL-low-balanced agent performs relatively bad

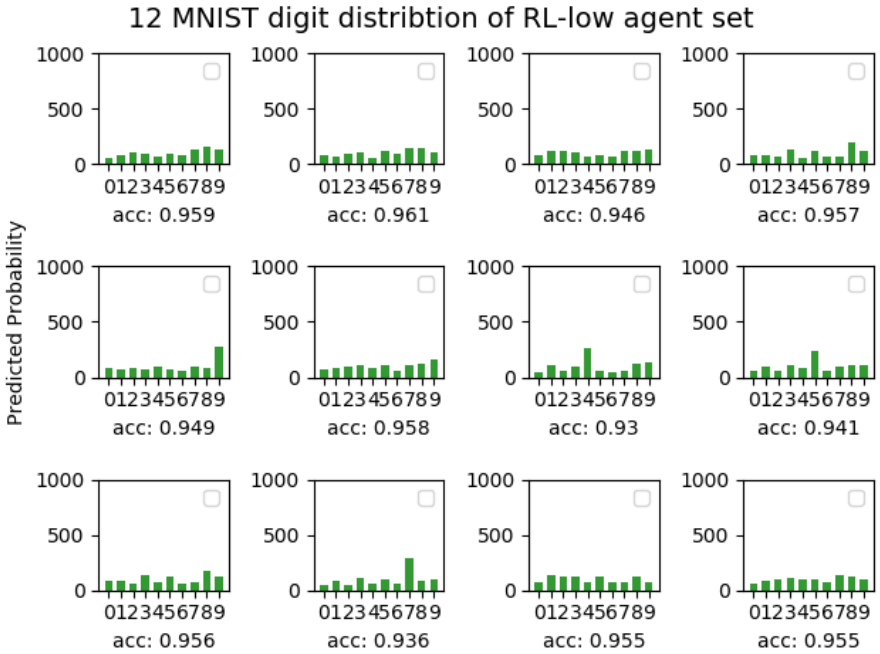

Figure 4.5: Digits Selection Distribution by RL-low agent. *x-axis:* digit class, *y-axis:* the selection frequency of each class

to its counterpart.

4.6 Effect of using exploration rate

We now move on to investigate the effect of training on exploration. In our experiment, the exploration rate decay with training steps. The exploration rate drops 0.0025 per step. We plot the exploration rate per each episode (number of trained classification networks). *Figure:* 4.8 shows that when the exploration drops and the agents learn from its policy, the accuracy drops along with the drop of exploration rate.

A straightforward explanation of this result is that the RL agent learns its biased policy and accumulate the bias selection.

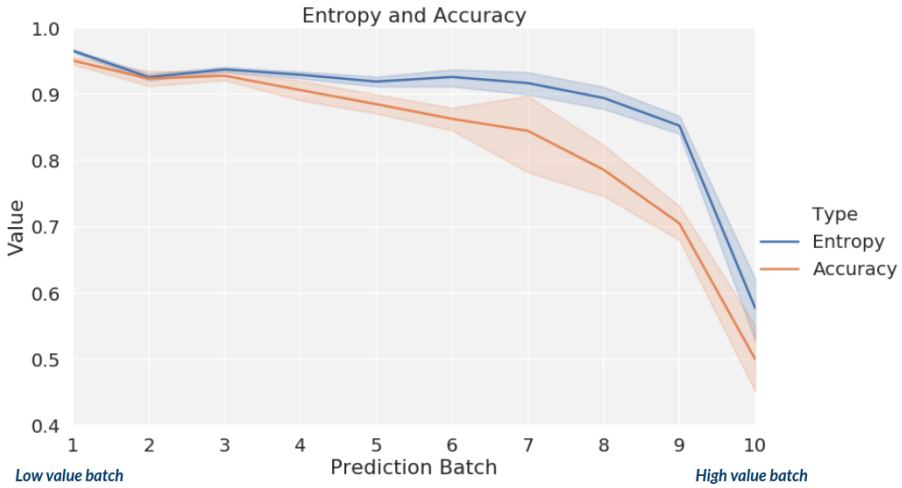

Figure 4.6: The relation of entropy and accuracy

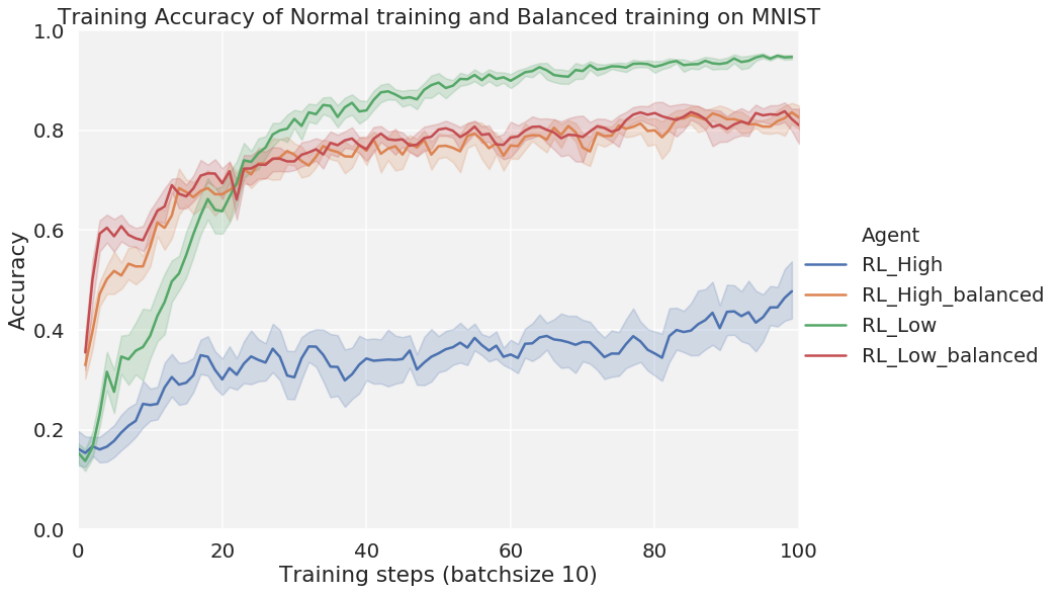

Figure 4.7

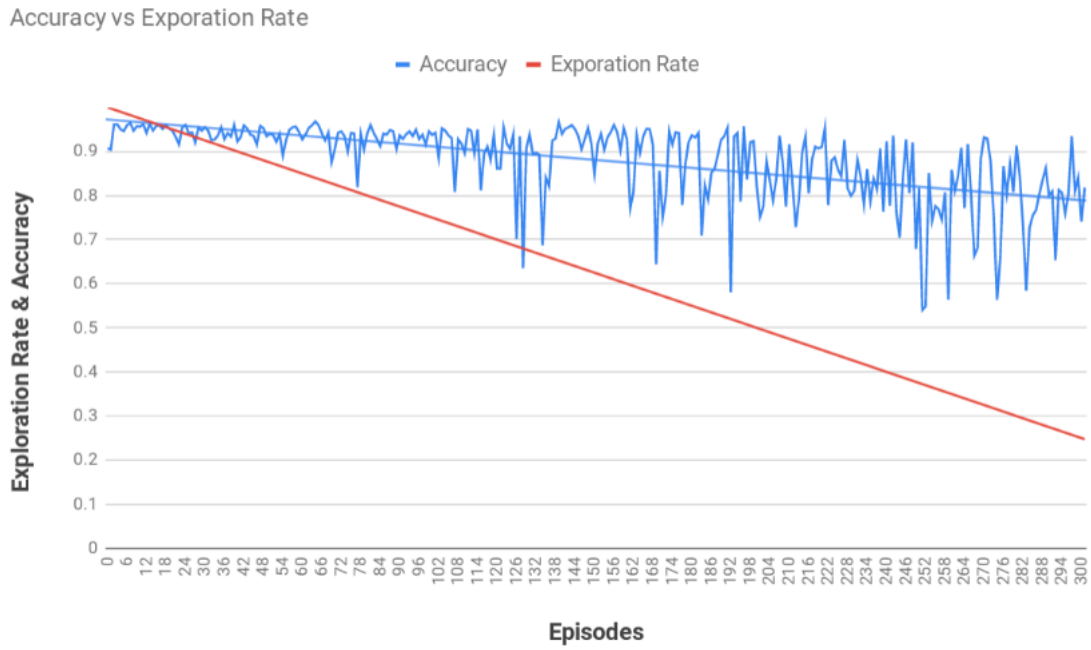
Accuracy vs Exporation Rate
Accuracy
Exporation Rate
Exploration Rate & Accuracy
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0 6 12 18 24 30 36 42 48 54 60 66 72 78 84 90 96 102 108 114 120 126 132 138 144 150 156 162 168 174 180 186 192 198 204 210 216 222 228 234 240 246 252 258 264 270 276 282 288 294 300
Episodes

Figure 4.8

5 Conclusion

We have proposed a novel training scheme to train a reinforcement learning agent to perform active learning. This method achieves a competitive result on the MNIST and EMNIST dataset. However, it fails to achieve a significant result on the CIFAR-10 dataset.

One reason could be due to the design of the training scheme as mentioned above, training a small batch of the CIFAR-10 dataset with a high number of epochs harms the classification network training leading to a lousy reward (accuracy) for the reinforcement learning agent to learn. On the other hand, the improvement of the classification accuracy becomes small when we reduce the numbers of training epoch. This also makes the RL agent hard to learn the active learning policy.

Another surprising result we have is that the RL-low agent performs better than the RL-top agent in all three datasets. This contradicts the reinforcement learning algorithm as the RL-top agent always selects image batch with the highest expected future return G_t. After we carried out some experiments, we find out the RL-top agent suffers from its biased selection policy. Hence, 1 or 2 digits with high probability have a dominated population of a batch. This, not surprisingly, damages the classification training and achieve bad terminal accuracy.

In future work, we plan to improve the performance of the algorithm on the CIFAR-10 dataset. This may need to reform and modify our existing algorithm to make it training compatible with more harder datasets. Besides, we plan to exploit the discarded unlabeled samples as the current algorithm does not use or learn any information from the discarded samples. We may achieve a significant improvement if we can exploit the information of the discarded samples.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *ArXiv*, abs/1603.04467, 2015.

[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *ArXiv*, abs/1708.05866, 2017.

[3] Franois Chollet. keras. https://github.com/fchollet/keras, 2015.

[4] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[5] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, V. Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, 2019.

[6] Yann Dauphin, Harm de Vries, and Yoshua Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. In *NIPS*, 2015.

[7] Meng Fang, Yuan Li, and Trevor Cohn. Learning how to active learn: A deep reinforcement learning approach. In *EMNLP*, 2017.

[8] Philip Häusser, Alexander Mordvintsev, and Daniel Cremers. Learning by association a versatile semi-supervised training method for neural networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 626–635, 2017.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[10] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[11] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[12] David D. Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *ICML*, 1994.

[13] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *SIGIR*, 1994.

[14] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[15] Martijn Otterlo Marco Wiering. *Reinforcement learning and markov decision processes*. Springer Berlin Heidelberg, 2012.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[17] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning. *CoRR*, abs/1712.00378, 2017.

[18] Nicholas D Roy and D. Archibald Mccallum. Toward optimal active learning through monte carlo estimation of error reduction. In *ICML 2001*, 2001.

[19] Tobias Scheffer, Christian Decomain, and Stefan Wrobel. Active hidden markov models for information extraction. In *IDA*, 2001.

[20] Nico Schlmer. Tuna. https://github.com/nschloe/tuna, 2019.

[21] H. Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *COLT*, 1992.

[22] Yanyao Shen, Hyokun Yun, Zachary Chase Lipton, Yakov Kronrod, and Anima Anandkumar. Deep active learning for named entity recognition. In *ICLR*, 2017.

[23] Mattew Slotkin. Accelerate machine learning with active learning, 2017.

[24] Matthijs T. J. Spaan. Partially observable markov decision processes. In *Reinforcement Learning*, 2012.

[25] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, 2001.

[26] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.