Learning how to Active Learn: A Deep Reinforcement Learning Approach

Meng Fang, Yuan Li, Trevor Cohn

The University of Melbourne

Presenter: Jialin Song

April 05, 2018
Overview

1. Introduction

2. Model

3. Algorithms

4. Numerical Experiments
Introduction: Active Learning

Annotation:

1. select a subset of data to annotate from a large unlabelled dataset (adding labels)
2. then we can train a supervised learning model $\phi$ (classifier)
3. we hope to maximize the accuracy of the classification model

Active learning:

- there is high cost annotating every sentence
- how to select raw data to add labels in order to maximize the accuracy of the classification model
- active learning becomes a sequential decision: as each sentence arrives, annotate it or not (our action)
Introduction: Active Learning

1 Annotation:

- select a subset of data to annotate from a large unlabelled dataset (adding labels)
Introduction: Active Learning

Annotation:

- select a subset of data to annotate from a large unlabelled dataset (adding labels)

- then we can train a supervised learning model $\phi$ (classifier)
Introduction: Active Learning

Annotation:

- select a subset of data to annotate from a large unlabelled dataset (adding labels)
- then we can train a supervised learning model $\phi$ (classifier)
- we hope to maximize the accuracy of the classification model
Introduction: Active Learning

1. Annotation:
   - select a subset of data to annotate from a large unlabelled dataset (adding labels)
   - then we can train a supervised learning model $\phi$ (classifier)
   - we hope to maximize the accuracy of the classification model

2. Active learning:
Introduction: Active Learning

1. Annotation:
   - select a subset of data to annotate from a large unlabelled dataset (adding labels)
   - then we can train a supervised learning model $\phi$ (classifier)
   - we hope to maximize the accuracy of the classification model

2. Active learning:
   - there is high cost annotating every sentence
Introduction: Active Learning

1. Annotation:
   - select a subset of data to annotate from a large unlabelled dataset (adding labels)
   - then we can train a supervised learning model $\phi$ (classifier)
   - we hope to maximize the accuracy of the classification model

2. Active learning:
   - there is high cost annotating every sentence
   - how to select raw data to add labels in order to maximize the accuracy of the classification model
Introduction: Active Learning

1. Annotation:
   - select a subset of data to annotate from a large unlabelled dataset (adding labels)
   - then we can train a supervised learning model \( \phi \) (classifier)
   - we hope to maximize the accuracy of the classification model

2. Active learning:
   - there is high cost annotating every sentence
   - how to select raw data to add labels in order to maximize the accuracy of the classification model
   - active learning becomes a sequential decision: as each sentence arrives, annotate it or not (our action)
Introduction: MDP

Markov Decision Process (MDP):

- A framework to model a sequential decision process.
- In each decision stage, the agent observes state variables ($s$) and takes an action ($a$) to maximize its current payoff.
- After taking the action, a reward associated with the action and state ($r(s,a)$) is generated, and the current state transitions to the next state.
- The agent aims to maximize the expected value of rewards over all stages.
Introduction: MDP

Markov Decision Process (MDP):

- a framework to model a **sequential** decision process
Introduction: MDP

Markov Decision Process (MDP):

- a framework to model a **sequential** decision process

- in each decision stage, agent observes state variables \( (s) \) and take a action \( (a) \) to maximize its current payoff
Introduction: MDP

Markov Decision Process (MDP):

- a framework to model a **sequential** decision process

- in each decision stage, agent observes state variables ($s$) and take a action ($a$) to maximize its current payoff

- after taking the action, a reward associated with the action and state ($r(s, a)$) is generated and current state transits to next state
Introduction: MDP

Markov Decision Process (MDP):

- a framework to model a **sequential** decision process
- in each decision stage, agent observes state variables \((s)\) and take a action \((a)\) to maximize its current payoff
- after taking the action, a reward associated with the action and state \((r(s,a))\) is generated and current state transits to next state
- agent aims maximizing the **expected value** of rewards over all stages
The dynamics of MDP can be modeled in Bellman equations.

Bellman equation 1: value function
\[ J(s) = \max_a \left\{ \bar{r}(s,a) + \alpha \sum_{s'} P_{ss'}(a) J(s') \right\} \]

Bellman equation 2 (more common!): Q-function
\[ Q(s,a) = \bar{r}(s,a) + \alpha \sum_{s'} P_{ss'}(a) \max_u Q(s',u) \]

where \( \bar{r}(s,a) \) is the expected reward, \( P_{ss'}(a) \) is the transition probability from state \( s \) to \( s' \), and \( \alpha \) is the discount of reward.
Introduction: Bellman Equation

1. The dynamics of MDP can be modeled in Bellman equations

   ◊ Bellman equation 1: value function

   \[ J(s) = \max_a \{ \bar{r}(s, a) + \alpha \sum_{s'} P_{ss'}(a) J(s') \} \]
   \[ a_s^* = \arg\max J(s) \]
The dynamics of MDP can be modeled in Bellman equations

- **Bellman equation 1: value function**
  \[ J(s) = \max_a \left\{ \bar{r}(s, a) + \alpha \sum_{s'} P_{ss'}(a) J(s') \right\} \]
  \[ a_s^* = \arg\max J(s) \]

- **Bellman equation 2 (more common!): Q-function**
  \[ Q(s, a) = \bar{r}(s, a) + \alpha \sum_{s'} P_{ss'}(a) \max_u Q(s', u) \]
  \[ a_s^* = \arg\max Q(s, a) \]
Introduction: Bellman Equation

1. The dynamics of MDP can be modeled in Bellman equations

   - **Bellman equation 1: value function**
     \[ J(s) = \max_a \left\{ \bar{r}(s, a) + \alpha \sum_{s'} P_{ss'}(a) J(s') \right\} \]
     \[ a_s^* = \arg\max J(s) \]

   - **Bellman equation 2 (more common!): Q-function**
     \[ Q(s, a) = \bar{r}(s, a) + \alpha \sum_{s'} P_{ss'}(a) \max_u Q(s', u) \]
     \[ a_s^* = \arg\max Q(s, a) \]

   where \( \bar{r}(s, a) \) is the expected reward, \( P_{ss'}(a) \) is the transition probability from state \( s \) to \( s' \), \( \alpha \) is the discount of reward
Q-Learning

1. If $P_{ss'}(a)$ is known, then solve the Bellman equations (VI/PI) to get the optimal policy. There is no need to 'learn'!!!
Q-Learning

1. If $P_{ss'}(a)$ is known, then solve the Bellmen equations (VI/PI) to get the optimal policy. There is no need to 'learn'!!!

2. If $P_{ss'}(a)$ is not known, then how to compute Q-function becomes a learning problem

Q-learning:

$Q_{t+1}(s_t, a_t) = (1 - \epsilon_t) Q_t(s_t, a_t) + \epsilon_t (\bar{r}(s_t, a_t) + \alpha \max_u Q_t(s_{t+1}, u))$

where $t$ is iteration and $\epsilon_t$ is the learning rate

In practice, above is useless: $|S| \times |A|$ is huge
Q-Learning

1. If $P_{ss'}(a)$ is known, then solve the Bellmen equations (VI/PI) to get the optimal policy. There is no need to 'learn'!!!

2. If $P_{ss'}(a)$ is not known, then how to compute Q-function becomes a learning problem.

3. Q-learning:

$$Q_{t+1}(s_t, a_t) = (1 - \epsilon_t)Q_t(s_t, a_t) + \epsilon_t(\bar{r}(s_t, a_t) + \alpha \max_u Q_t(s_{t+1}, u))$$

where $t$ is iteration and $\epsilon_t$ is the learning rate.

In practice, above is useless: $|S| \times |A|$ is huge.
Q-Learning

1. If $P_{ss'}(a)$ is known, then solve the Bellmen equations (VI/PI) to get the optimal policy. There is no need to 'learn'!!!

2. If $P_{ss'}(a)$ is not known, then how to compute Q-function becomes a learning problem

3. Q-learning:

   $Q_{t+1}(s_t, a_t) = (1 - \epsilon_t)Q_t(s_t, a_t) + \epsilon_t(\bar{r}(s_t, a_t) + \alpha \max_u Q_t(s_{t+1}, u))$
Q-Learning

1. If $P_{s's'}(a)$ is known, then solve the Bellmen equations (VI/PI) to get the optimal policy. There is no need to 'learn'!!!

2. If $P_{s's'}(a)$ is not known, then how to compute Q-function becomes a learning problem.

3. Q-learning:
   \[
   Q_{t+1}(s_t, a_t) = (1 - \epsilon_t)Q_t(s_t, a_t) + \epsilon_t(\bar{r}(s_t, a_t) + \alpha \max_u Q_t(s_{t+1}, u))
   \]
   \[
   \text{where} \ t \ \text{is iteration and} \ \epsilon_t \ \text{is the learning rate}
   \]
Q-Learning

1. If $P_{ss'}(a)$ is known, then solve the Bellmen equations (VI/PI) to get the optimal policy. There is no need to 'learn'!!!

2. If $P_{ss'}(a)$ is not known, then how to compute Q-function becomes a learning problem

3. Q-learning:

   - $Q_{t+1}(s_t, a_t) = (1 - \epsilon_t)Q_t(s_t, a_t) + \epsilon_t(\bar{r}(s_t, a_t) + \alpha \max_u Q_t(s_{t+1}, u))$

   - where $t$ is iteration and $\epsilon_t$ is the learning rate

   - In practice, above is useless: $|S| \times |A|$ is huge
Deep Q-Learning

Deep Q-learning:

- **Input:** state $s$, action $a$, reward $r(s,a)$, state transition $s'$
- **Output:** approximation of Q-function: $f_\theta(s,u)$
- **Loss function minimization:**
  $$\min_\theta \left\{ \frac{1}{2} \left(f_\theta_t(s_t,a_t) - \bar{r}(s_t,a_t) - \alpha \max_u f_\theta_t(s_{t+1},u) \right)^2 \right\}$$
Deep Q-Learning

Deep Q-learning:

- use the output of a DNN parametrized by $\theta$, i.e., $f_\theta(s, u)$ to approximate $Q(s, a)$:
Deep Q-Learning

Deep Q-learning:

- use the output of a DNN parametrized by $\theta$, i.e., $f_\theta(s, u)$ to approximate $Q(s, a)$:

- input: state $s$, action $a$, reward $r(s, a)$, state transition $s'$
Deep Q-Learning

Deep Q-learning:

- use the output of a DNN parametrized by $\theta$, i.e., $f_\theta(s, u)$ to approximate $Q(s, a)$:

- input: state $s$, action $a$, reward $r(s, a)$, state transition $s'$

- output: approximation of Q-function: $f_\theta(s, u)$
Deep Q-Learning

Deep Q-learning:

- use the output of a DNN parametrized by $\theta$, i.e., $f_\theta(s, u)$ to approximate $Q(s, a)$:

- input: state $s$, action $a$, reward $r(s, a)$, state transition $s'$

- output: approximation of Q-function: $f_\theta(s, u)$

- the loss function minimization

$$\min_\theta \left\{ \frac{1}{2} \left( f_\theta(s_t, a_t) - \bar{r}(s_t, a_t) - \alpha \max_u f_\theta(s_{t+1}, u) \right)^2 \right\}$$
sentence $x_i$ from an unlabelled dataset arrives one by one
Model Active Learning as MDP

1. sentence $x_i$ from an unlabelled dataset arrives one by one

2. for each arriving sentence, agent decides whether to annotate it or not (binary action)
Model Active Learning as MDP

1. sentence $x_i$ from an unlabelled dataset arrives one by one

2. for each arriving sentence, agent decides whether to annotate it or not (binary action)

3. if agent annotates it, then the annotated set (labelled set) gets expanded, and the classifier $\phi$ is re-trained and updated
Model Active Learning as MDP

1. sentence $x_i$ from an unlabelled dataset arrives one by one

2. for each arriving sentence, agent decides whether to annotate it or not (binary action)

3. if agent annotates it, then the annotated set (labelled set) gets expanded, and the classifier $\phi$ is re-trained and updated

4. evaluate the updated classifier on a separate independent dataset, get the test accuracy (reward)
Model Active Learning as MDP

1. sentence $x_i$ from an unlabelled dataset arrives one by one

2. for each arriving sentence, agent decides whether to annotate it or not (binary action)

3. if agent annotates it, then the annotated set (labelled set) gets expanded, and the classifier $\phi$ is re-trained and updated

4. evaluate the updated classifier on a separate independent dataset, get the test accuracy (reward)

5. next sentence arrives
Model Active Learning as MDP

State (s): comprised of 2 parts:

1. Input set sentence: \( x_i \) (encoded using a CNN, \( h_c \))
2. Trained classification model: \( \phi \) (encoded using a CNN, \( h_e \))

Action (a):

- \( a_i = 1 \): annotate \( x_i \)
- \( a_i = 0 \): not annotate \( x_i \)

Reward (r):

- Evaluate the classification model on a held-out set after the action \( a \) is taken and get the test accuracy.
Model Active Learning as MDP

1. State (s): comprised of 2 parts:
   - input sentence: $x_i$ (encoded using a CNN, $h_c$)
Model Active Learning as MDP

- **State (s)**: comprised of 2 parts:
  - input sentence: $x_i$ (encoded using a CNN, $h_c$)
  - trained classification model $\phi$ (encoded using a CNN, $h_e$)

- **Action (a)**:
  - $a_i = 1$: annotate $x_i$
  - $a_i = 0$: not annotate $x_i$

- **Reward (r)**: evaluate the classification model on a held-out set after the action is taken and get the test accuracy
Model Active Learning as MDP

1. State (s): comprised of 2 parts:
   - input sentence: $x_i$ (encoded using a CNN, $h_c$)
   - trained classification model $\phi$ (encoded using a CNN, $h_e$)

2. Action (a):
Model Active Learning as MDP

1. State (s): comprised of 2 parts:
   - input sentence: $x_i$ (encoded using a CNN, $h_c$)
   - trained classification model $\phi$ (encoded using a CNN, $h_e$)

2. Action (a):
   - $a_i = 1$: annotate $x_i$
Model Active Learning as MDP

1. State (s): comprised of 2 parts:
   - input sentence: $x_i$ (encoded using a CNN, $h_c$)
   - trained classification model $\phi$ (encoded using a CNN, $h_e$)

2. Action (a):
   - $a_i = 1$: annotate $x_i$
   - $a_i = 0$: not annotate $x_i$
Model Active Learning as MDP

1. State (s): comprised of 2 parts:
   - Input sentence: $x_i$ (encoded using a CNN, $h_c$)
   - Trained classification model $\phi$ (encoded using a CNN, $h_e$)

2. Action (a):
   - $a_i = 1$: annotate $x_i$
   - $a_i = 0$: not annotate $x_i$

3. Reward (r):
Model Active Learning as MDP

1. State (s): comprised of 2 parts:
   - input sentence: $x_i$ (encoded using a CNN, $h_c$)
   - trained classification model $\phi$ (encoded using a CNN, $h_e$)

2. Action (a):
   - $a_i = 1$: annotate $x_i$
   - $a_i = 0$: not annotate $x_i$

3. Reward (r):
   - evaluate the classification model on a held-out set after the action $a$ is taken and get the test accuracy
An Value Iteration Q-learning Algorithm

Algorithm 1 Learn an active learning policy

Input: data $\mathcal{D}$, budget $B$
Output: $\pi$

1: for episode = 1, 2, ..., $N$ do
2: $\mathcal{D}_i \leftarrow \emptyset$ and shuffle $\mathcal{D}$
3: $\phi \leftarrow$ Random
4: for $i \in \{0, 1, 2, \ldots, |\mathcal{D}|\}$ do
5: Construct the state $s_i$ using $x_i$
6: The agent makes a decision according to
7: if $a_i = 1$ then action: annotate
8: Obtain the annotation $y_i$
9: $\mathcal{D}_i \leftarrow \mathcal{D}_i + (x_i, y_i)$
10: Update model $\phi$ based on $\mathcal{D}_i$
11: end if
12: Receive a reward $r_i$ using held-out set
13: if $|\mathcal{D}_i| = B$ then test classifier on a separate set
14: Store $(s_i, a_i, r_i, \text{Terminate})$ in $\mathcal{M}$
15: Break
16: end if
17: Construct the new state $s_{i+1}$
18: Store transition $(s_i, a_i, r_i, s_{i+1})$ in $\mathcal{M}$
19: Sample random minibatch of transitions $\{(s_j, a_j, r_j, s_{j+1})\}$ from $\mathcal{M}$, and perform gradient descent step on $L(\theta)$
20: Update policy $\pi$ with $\theta$
21: end for
22: end for
23: return the latest policy $\pi$
Important Step 1

\[ \text{for } i \in \{0, 1, 2, \ldots, |D|\} \text{ do} \]
\[ \text{Construct the state } s_i \text{ using } x_i \]
\[ \text{The agent makes a decision according to } \]
\[ a_i = \arg \max Q^\pi (s_i, a) \]
\[ \text{if } a_i = 1 \text{ then} \]
\[ \text{Obtain the annotation } y_i \]
\[ D_l \leftarrow D_l + (x_i, y_i) \]
\[ \text{Update model } \phi \text{ based on } D_l \]
\[ \text{end if} \]
\[ \text{Receive a reward } r_i \text{ using held-out set} \]
Important Step 2

Construct the new state $s_{i+1}$
Store transition $(s_i, a_i, r_i, s_{i+1})$ in $\mathcal{M}$
Sample random minibatch of transitions $\{(s_j, a_j, r_j, s_{j+1})\}$ from $\mathcal{M}$, and perform gradient descent step on $\mathcal{L}(\theta)$
Update policy $\pi$ with $\theta$

end for
Remarks on the Q-learning algorithm:

```plaintext
for episode = 1, 2, ..., N do
```

1. **Remarks on the Q-learning algorithm:**
for episode $= 1, 2, \ldots, N$ do

Remarks on the Q-learning algorithm:

- input: unlabelled dataset $D$
for episode = 1, 2, ..., N do

Remarks on the Q-learning algorithm:

- input: unlabelled dataset $D$
- output: a series of actions ($a_i$): policy $\pi$
Relaxation 1: Transfer Policy

1. train annotation policy $\pi$ in source language (e.g., English) and transfer it to low-source target language
Relaxation 1: Transfer Policy

1. train annotation policy $\pi$ in source language (e.g., English) and transfer it to low-source target language

Algorithm 2: Active learning by policy transfer

Input: unlabelled data $\mathcal{D}$, budget $\mathcal{B}$, policy $\pi$

Output: $\mathcal{D}_l$

1. $\mathcal{D}_l \leftarrow \emptyset$
2. $\phi \leftarrow \text{Random}$
3. for $|\mathcal{D}_l| \neq \mathcal{B}$ and $\mathcal{D}$ not empty do
4. Randomly sample $x_i$ from the data pool $\mathcal{D}$ and construct the state $s_i$
5. The agent chooses an action $a_i$ according to $a_i = \text{arg max } Q(\pi)_{s_i, a}$
6. if $a_i = 1$ then
7. Obtain the annotation $y_i$
8. $\mathcal{D}_l \leftarrow \mathcal{D}_l + (x_i, y_i)$
9. Update model $\phi$ based on $\mathcal{D}_l$
10. end if
11. $\mathcal{D} \leftarrow \mathcal{D} \setminus x_i$
12. Receive a reward $r_i$ using held-out set
13. Update policy $\pi$
14. end for
15. return $\mathcal{D}_l$
Relaxation 2: Transfer Model and Policy

1. train a classification model $\phi$ and annotation policy $\pi$ in source language (e.g., English) and transfer both to low-source target language.
Relaxation 2: Transfer Model and Policy

1. train a classification model $\phi$ and annotation policy $\pi$ in source language (e.g., English) and transfer both to low-source target language.

2. this relaxation is more like a test and implementation procedure.

---

**Algorithm 3** Active learning by policy and model transfer, for ‘cold-start’ scenario

**Input:** unlabelled data $D$, budget $B$, policy $\pi$, model $\phi$ (trained dataset), annotation policy $\pi$ (trained policy)

**Output:** $D_t$

1. $D_t \leftarrow \emptyset$
2. for $|D_t| \neq B$ and $D$ not empty do
3. Randomly sample $x_i$ from the data pool $D$ and construct the state $s_i$
4. The agent chooses an action $a_i$ according to $a_i = \arg \max Q^\pi(s_i, a)$
5. if $a_i = 1$ then
6. $D_t \leftarrow D_t + (x_i, -)$ annotate based on $\phi$
7. end if
8. $D \leftarrow D \setminus x_i$
9. end for
Numerical Experiments

A couple of numerical experiments show that the newly proposed active learning approach by deep Q-learning works better than some existing active learning methods such as uncertainty sampling and random sampling.

![Graph showing F1 score vs number of labelled sentences for different languages and methods.](image)
Thank You!

.....Question?