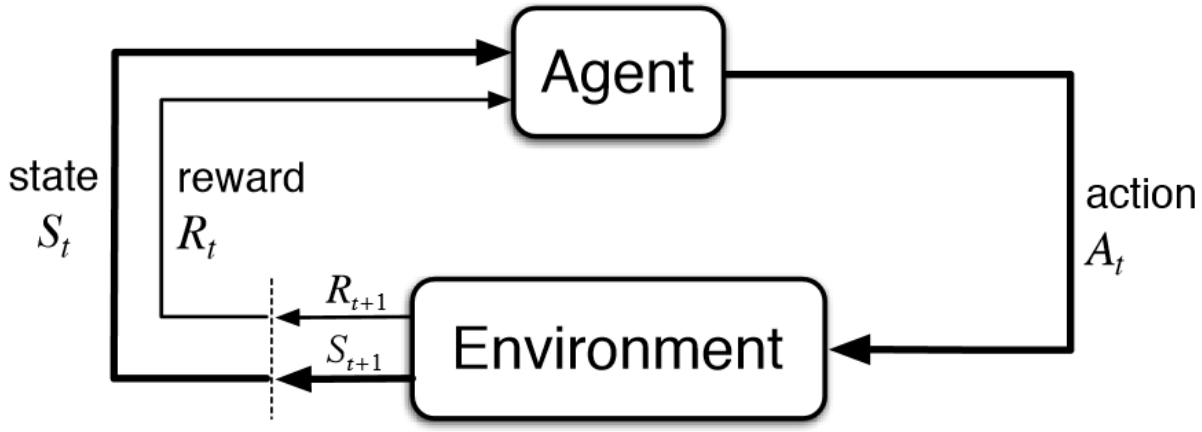
# Reinforcement Learning

Reinforcement Learning is a computational approach of learning from interaction with an environment. Both supervised and reinforcement learning use mapping between input and output, in supervised learning where the feedback provided to the agent is the correct set of actions for performing a task, reinforcement learning uses rewards and punishments as signals for positive and negative behavior. In RL the goal is to find a suitable action model that would maximize the total cumulative reward of the agent.

## Key Elements of Reinforcement Learning

1. **Policy ($\pi$):** Are mappings from states to actions that define agent's behaviour. It helps to decide the next action based on the current state.
2. **Reward (R):** Is an immediate return given to an agent when he or she performs a specific action or task.It is the primary basis for changing the policy. Environment sends a reward at each time step. The Reinforcement agent tries to maximize the reward. Rewards are given directly by the environment but the values must be estimated and re-estimated from the sequence of observations an agent makes over its entire lifetime.
3. **Value Function:** Assigns value to the states and specifies what is good in the long run vs reward which is an immediate signal. Value of a state is the reward the agent can expect starting from that state. Efficiently estimating values is the most important component of all RL algorithms.
4. **Model :** It mimics the behavior of the environment and allows inference about how the environment might behave. Given a state and action, the model might predict the resultant next state and next reward.
5. **Agent:** It is an assumed entity which performs actions in an environment to gain some reward.
6. **State (s):** State refers to the current situation returned by the environment.
7. **Value (V):** It is expected long-term return with discount, as compared to the short-term reward.

8. **Q value or action value (Q):** Q value is quite similar to value. The only difference between the two is that Q value takes an additional parameter as a current action. It is an estimation of how good it is to take action A at state the S.




## Exploration vs Exploitation trade-off

In order to build an optimal policy, the agent faces the dilemma of exploring new states while maximizing its overall reward at the same time called as **Exploration vs Exploitation** trade-off. To balance both, the best overall strategy may involve short term sacrifices. Therefore, the agent should collect enough information to make the best overall decision in the future.

## Markov Decision Processes(MDPs)

These are mathematical frameworks to describe an environment in RL and almost all RL problems can be formulated using MDPs. A MDP consists of a set of finite environment states S, a set of possible actions A(s) in each state, a real valued reward function R(s) and a transition model P(s', s | a). However, real world environments are more likely to lack any prior knowledge of environment

dynamics. Model-free RL methods are used in such cases.

## Q- Learning

It is a commonly used model-free approach which uses Q-values (also called action values) to iteratively improve the behavior of the learning agent. Q-values are defined for states and actions. Q(S,A) is an estimation of how good it is to take the action A at the state S. This estimation of Q(S,A) will be iteratively computed using the following Temporal Difference or TD-Update rule which is the core of the Q-learning algorithm

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

## Multi-Armed Bandit Problem

The term "multi-armed bandit" comes from a hypothetical experiment where a person must choose between multiple actions (i.e. slot machines) each with an unknown payout. The goal is to determine the best or most profitable outcome through a series of choices. At the beginning of the experiment, when odds and payouts are unknown, the gambler must determine which machine to pull, in which order and how many times. This is the “multi-armed bandit problem.”

The multi-armed bandit solution is a smarter version of A/B testing that uses machine learning algorithms to dynamically allocate traffic to variations that are performing well, while allocating less traffic to variations that are underperforming.

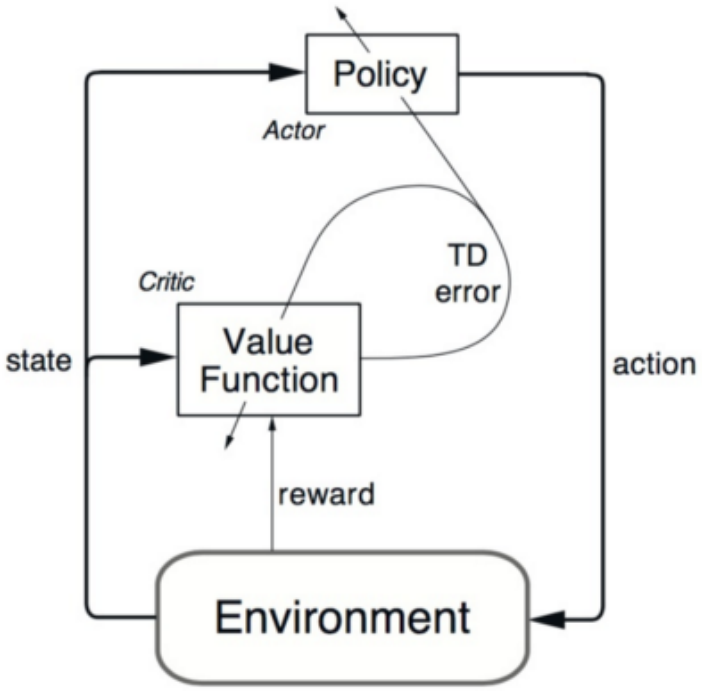
# Reinforcement Learning algorithms

**Q-learning** and State-Action-Reward-State-Action(**SARSA**) are two commonly used model-free RL algorithms. They differ in terms of their exploration strategies but their exploitation strategies are similar. While Q-learning is an off-policy

method in which the agent learns the value based on action a* derived from another policy, SARSA(state-action-reward-state-action) is an on-policy method where it learns the value based on its current action derived from its current policy. These two methods are simple to implement but lack generality as they do not have the ability to estimate values for unseen states.

This can be overcome by more advanced algorithms such as Deep Q-Networks(DQNs**)** which use Neural Networks to estimate Q-values. But DQNs can only handle discrete, low-dimensional action spaces.

**Deep Deterministic Policy Gradient(DDPG)** is a model-free, off-policy, actor-critic algorithm that tackles this problem by learning policies in high dimensional, continuous action spaces. The figure below is a representation of **actor-critic** architecture.



# Large Scale Open Dataset for and Pipeline for Bandit Algorithms

## Data Generation Process

1. Observes X(the context vector like user visit)
2. A policy **(π)** selects an action A (e.g. an item)
3. Observes Y( e.g. click indicator)

A policy interacts with the environments and produces log data.

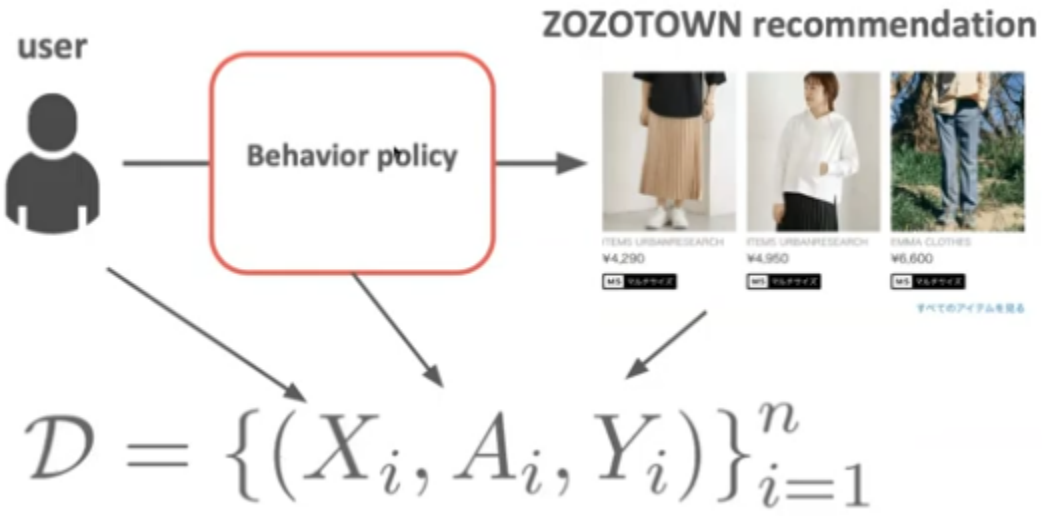
**Logged Bandit Feedback**

$$\mathcal{D} = \{(X_i, A_i, Y_i)\}_{i=1}^{n}$$

$A_i \sim \pi_b(a \mid X_i)$ — action choice by behavior policy

$Y_i = Y_i(A_i)$ — observed reward

,




$$\mathcal{D} = \{(X_i, A_i, Y_i)\}_{i=1}^{n}$$

Logged Bandit Feedback collected by a behaviour or past policy can be used to estimate the policy value of a new policy.

The off-policy methods evaluate or improve a policy different from that used to generate the data. In off-policy evaluation we aim to estimate the following policy value of a new policy i.e the expected reward obtained by running $\pi_e$ on a real system.

$$V(\pi_e) := \mathbb{E}_{(Y(\cdot),X)}[\sum_{a=0}^{m} Y(a)\pi_e(a \mid X)]$$

**Benefits of Off-Policy Evaluation**

1. Helps to avoid deploying poor performing policies.
2. Identify promising new policies among many candidates.

## Different estimators in Off-Policy Evaluation are as follows:

1. **Direct Method**

**DM first estimates the expected reward and uses it to estimate the policy value**

$$\hat{V}_{DM}(\pi_e; \mathcal{D}) = \mathbb{E}_n[\sum_{a=0}^{m} \pi(a \mid X_i)\, \hat{\mu}(X_i, a)]$$

estimated expected reward

- High bias when the model is mis-specified
- Low variance

$$\mathbb{E}[Y(a) \mid X = x] \approx \hat{\mu}(x, a)$$

2. **Inverse Probability Rating**

**IPW re-weighs observed rewards by importance weights**

$$\hat{V}_{IPW}(\pi_e; \mathcal{D}) = \mathbb{E}_n \left[ Y_i \frac{\pi_e(A_i \mid X_i)}{\pi_b(A_i \mid X_i)} \right]$$

importance weight

- Consistent
- High variance when old and new policies are largely different

3. **Doubly Robust**

**DR uses DM as a baseline and applies IPW to shifted rewards**

$$\hat{V}_{DR}(\pi_e; \mathcal{D}) = \hat{V}_{DM}(\pi_e; \mathcal{D}) + \mathbb{E}_n \left[ (Y_i - \hat{\mu}(X_i, A_i)) \frac{\pi_e(A_i|X_i)}{\pi_b(A_i|X_i)} \right]$$

baseline

weighted shifted reward

- Consistent
- Locally Efficient

$$\mathbb{E}[Y(a) \mid X = x] \approx \hat{\mu}(x, a)$$

Issues with the current experimental procedure is that experiments in every Off-Policy Evaluation paper rely on synthetic unrealistic data or real unpublished data.

The goal of the open bandit project is to enable realistic and reproducible experiments on bandit algorithms and Off-Policy Evaluation.

## Overview of Open Bandit Dataset

### Schema of Open Bandit Dataset

$A$ $\pi_b(\cdot|X)$ $Y$ $X$

| timestamp | item_id | position | propensity _score | click indicator | features | ... |
|---|---|---|---|---|---|---|
| 2019-11-xx | 25 | 1 | 0.0002 | 0 | e2500f3f | ... |
| 2019-11-xx | 32 | 2 | 0.043 | 1 | 7c414ef7 | ... |
| 2019-11-xx | 11 | 3 | 0.167 | 0 | 60bd4df9 | ... |
| 2019-11-xx | 40 | 1 | 0.0011 | 0 | 7c20d9b5 | ... |
| ... | ... | ... | ... | ... | ... | ... |

### Essential Features of Open Bandit Dataset

- over 25M records collected by online experiments of bandit algorithms on a large-scale fashion e-commerce (*ZOZOTOWN*)
- logged bandit feedback collected by *multiple* bandit policies
  - *Uniform Random* (fixed)
  - *Bernoulli Thompson Sampling* (pre-trained before collection)

# Protocol for the Evaluation of Off-Policy Evaluation with Open Bandit Dataset

1. Prepare *two* logged bandit feedback data collected by different policies

$$\mathcal{D}^{(1)} = \left\{ \left( X_i^{(1)}, A_i^{(1)}, Y_i^{(1)} \right) \right\}_{i=1}^{n}$$

collected by $\pi^{(1)}$

$$\mathcal{D}^{(2)} = \left\{ \left( X_i^{(2)}, A_i^{(2)}, Y_i^{(2)} \right) \right\}_{i=1}^{n}$$

collected by $\pi^{(2)}$

---

2. Regard one policy as an *evaluation policy* and the other as a *behavior policy*. Then, estimate the performance of the evaluation policy by OPE

$$V(\pi^{(1)}) \approx \hat{V}(\pi^{(1)}; \mathcal{D}^{(2)})$$

$\pi^{(1)}$ : evaluation policy $\pi^{(2)}$ : behavior policy

- The task here is to evaluate the accuracy of $\hat{V}$

3. Regard the *on-policy estimation* of the policy value of the evaluation policy as the ground-truth policy value

$$V(\pi^{(1)}) = \mathbb{E}_{n^{(1)}}[Y^{(1)}]$$

we can do this on-policy estimation because we have $\mathcal{D}^{(1)}$ in our data

4. Compare the estimated policy value with the ground-truth to evaluate the OPE estimator, for example, using the *relative estimation error*

$$\text{relative estimation error of } \hat{V} = \left| \frac{\hat{V}\left(\pi^{(1)}; \mathcal{D}^{(2)}\right) - V\left(\pi^{(1)}\right)}{V\left(\pi^{(1)}\right)} \right|$$

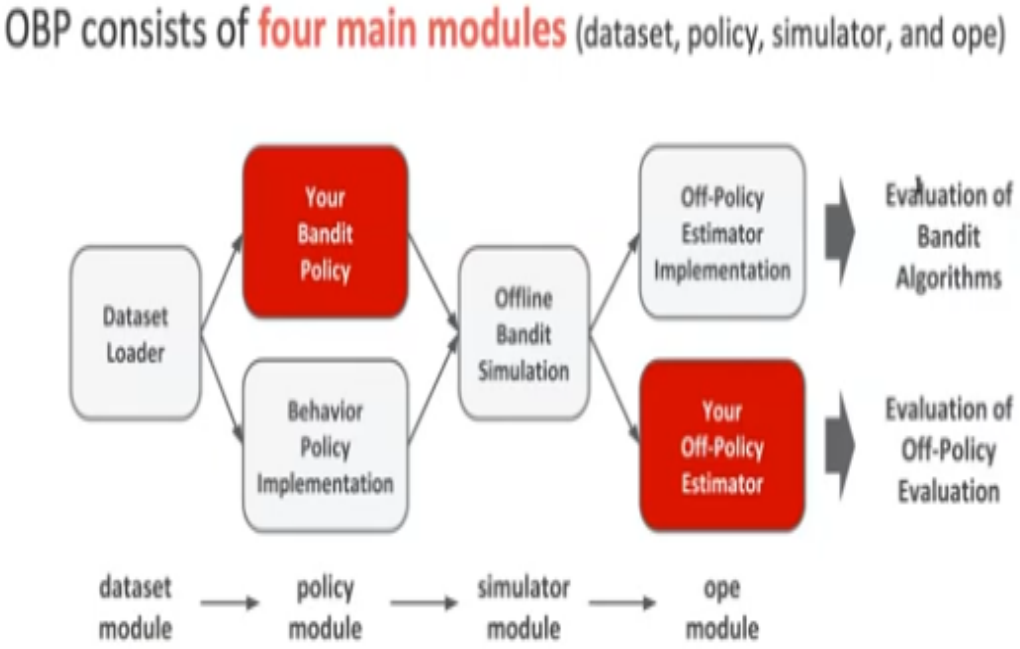
By applying this procedure to several estimators, we can compare them

## Comparison of currently available large scale bandit datasets

| | **Criteo Data** (Lefortier et al. 2016) | **Yahoo! Data** (Li et al. 2010) | **Open Bandit Dataset** (ours) |
|---|---|---|---|
| **Domain** | Display Advertising | News Recommendation | Fashion E-Commerce |
| **#Data** | >= 103M | >= 40M | >= 26M (will increase) |
| **#Behavior Policies** | 1 | 1 | 2 (will increase) |
| **Random A/B Test Data** | ✗ | ✔ | ✔ |
| **Behavior Policy Code** | ✗ | ✗ | ✔ |
| **Evaluation of Bandit Algorithms** | ✔ | ✔ | ✔ |
| **Evaluation of OPE** | ✗ | ✗ | ✔ |
| **Pipeline Implementation** | ✗ | ✗ | ✔ |

Open Bandit Pipeline implemented to streamline and standardize experiments in Off-Policy Evaluation.

## Structure of Open Bandit Pipeline




## Benchmark comparison on basic OPE estimators using Relative-Estimation errors of estimators.

| | Campaigns | | |
|---|---|---|---|
| **Estimators** | **All** | **Men's** | **Women's** |
| DM | 0.23879 [0.22998, 0.24988] | 0.24155 [0.22656, 0.25592] | 0.22884 [0.22224, 0.23423] |
| IPW | 0.03477 [0.01147, 0.06592] | 0.09806 [0.07485, 0.12151] | 0.03252 [0.01708, 0.04912] |
| SNIPW | 0.03381 [0.01005, 0.06662] | 0.08153 [0.05677, 0.10592] | 0.03179 [0.01562, 0.04825] |
| DR | 0.03487 [0.01094, 0.06784] | 0.08528 [0.06186, 0.10876] | 0.03224 [0.01605, 0.04843] |

DM- Direct Method

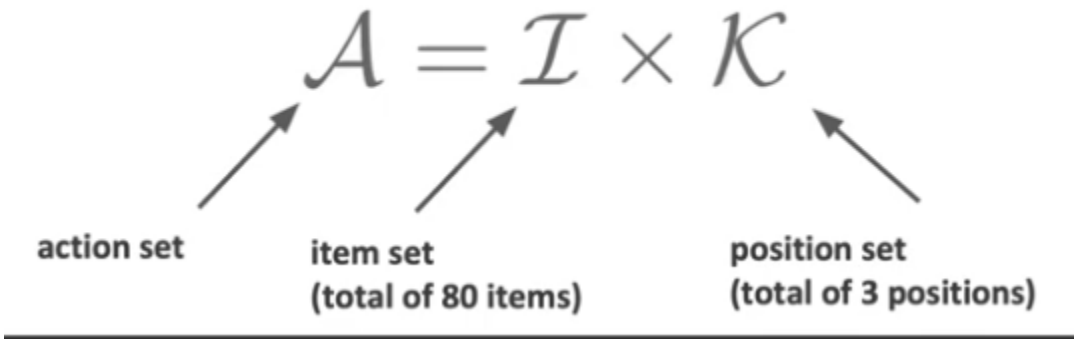
IPW- Inverse Probability Rating

SNIPW- Self-Normalized Inverse Probability Weighting

DR- Doubly Robust

## Limitations of the project-

1. It is assumed that the click of an item(reward) depends on that item and position.




That is the action set can be decomposed into an item set and a position set. It ignores the interactions among items in the same list.

2. The current data contains only context free policies and does not use any contextual information.

# Distributed Asynchronous Deep Reinforcement Learning Framework for Recommender Systems

### Why is distributed RL better?

1. Increasingly powerful mobile devices.
2. Enormous user private data.
3. Waste of resources and risky to rely on the central server.
4. Scalability and the frequency of model update
5. Dynamic and interactive environment
6. Long term user satisfaction due to better response time.

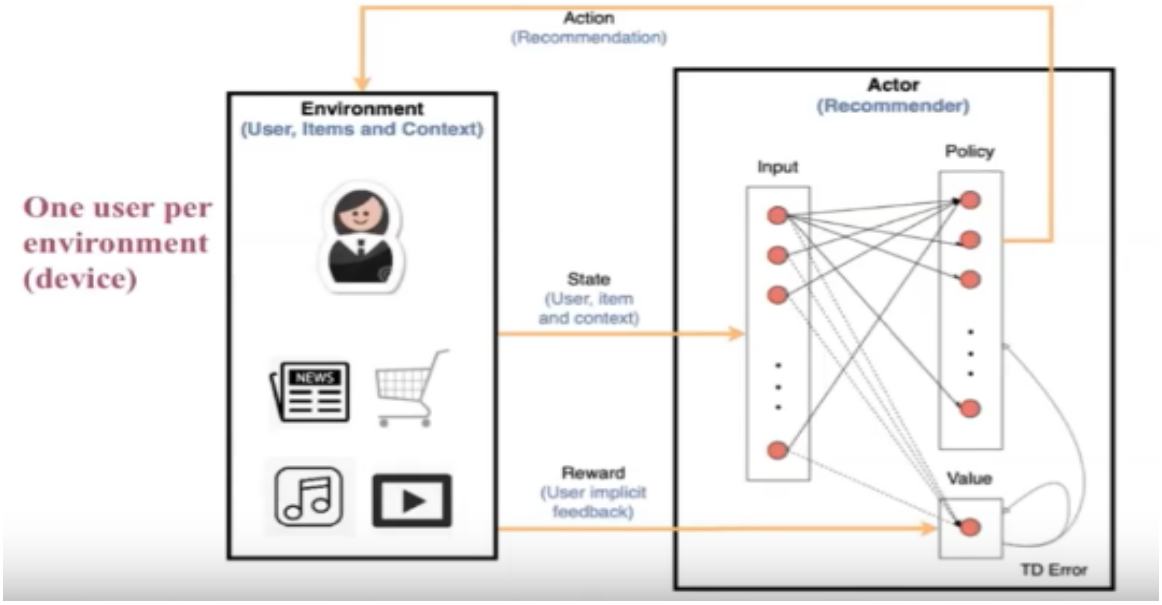
## Features of Distributed Asynchronous Deep Reinforcement Learning Framework

1. Model the item recommendation/ rating prediction as classification problem.
2. Use advantage actor critic reinforcement learning method(A3C)
3. Keep user data on the local device
4. Training in on the local devices coordinated by the central server
5. Share the gradient of losses with the central server.
6. Updated in an asynchronous manner i.e. the user device can draw in or leave the training process anytime regardless of the online\offline status of the device.

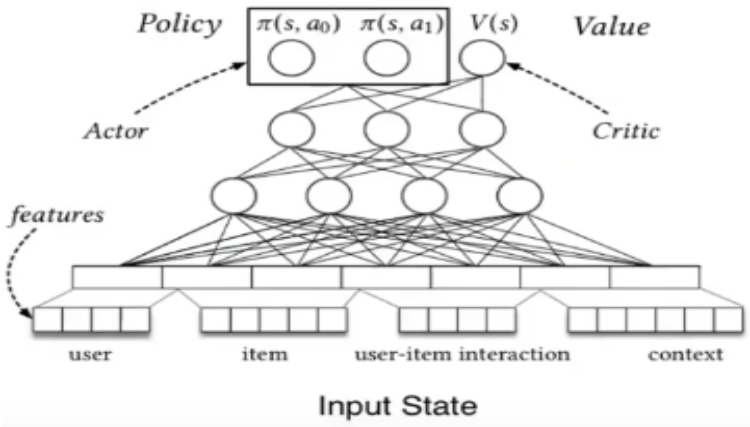
# Item Recommendation as reinforcement learning problem

For recommender systems the environment is the recommendation platform where the users are requesting information in a certain context. For instance recommending news articles on a news website. The user, the candidate item and the context is saved in state. And the state is sent to the actor which is the recommender. The recommender learns a policy and induces an action which is recommending an item. The user receives the recommendation and gives some feedback whether he clicks or likes the item the feedback in the update state and the context are sent back to the recommender.

The recommender uses the feedback to learn the value function and helps to improve the recommendation policy. The loop continues until the user leaves the platform. Since the interaction will continue for multiple rounds the recommender learns to recommend items that can potentially generate more reward in the long run. Each user has its unique environment as it has preference and clicking behaviour. When the user data is not enough for model training we need a central server to coordinate the training between all user devices.



# Network Structure and Features

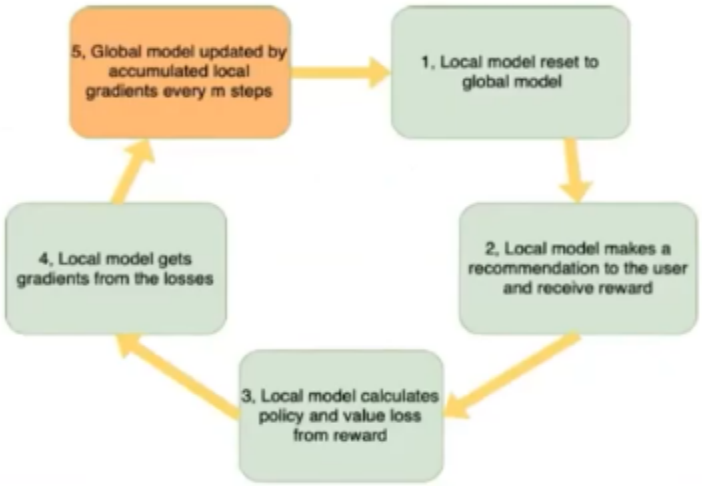



Input State

- Input state represents the relevancy of a user-item pair.
  - User demographic: gender, age, etc
  - Item details: title, description, popularity, etc
  - User-item interaction: user's browsing history, etc
  - Context: time, location, etc
- Two headed neural network, one head for Policy and one for Value.
- The action space is recommend the item or not (or ratings). Candidate items are not hard-coded into the action space.

# System Architecture

- Central server keeps a global model
- Local devices keep a copy of the global model
- Synchronous every $m$ steps
- Reward function reflects the recommendation accuracy
- Loss functions:




$$L_v = \sum (R_t - V(s_t; \theta_v))^2$$

$$L_p = -\log(\pi(a_t|s_t;\theta))A(s_t, a_t; \theta, \theta_v) - \beta H(\pi(s_t;\theta))$$

# Comparison to State-of-the-art

- Compared to the following methods:
  - LinUCB, a MAB-based approach as presented in [1].
  - DQN and DDQN, similar to the Drn work in [2].
  - A2C-D in its original distributed synchronous setting.
  - A2C-F, in federated-liked setting, similar to DADRL, updated in synchronous style.
- Evaluation metric:
  - Click through rate (CTR) = $\frac{\text{Number of clicks}}{\text{Number of recommendation}}$
- Dataset:
  - A subset of the Outbrain dataset, collected for news click prediction purpose.

| Users | Items | Interactions | Avg Session Length |
|---|---|---|---|
| 22,713 | 2,302 | 441,213 | 19.42 |

# Comparison to State-of-the-art

Training

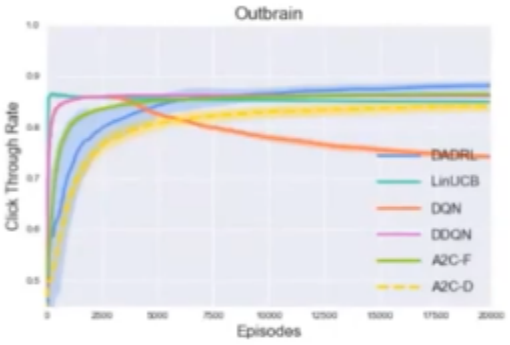



Figure. Global average CTR vs number of training episodes for Outbrain

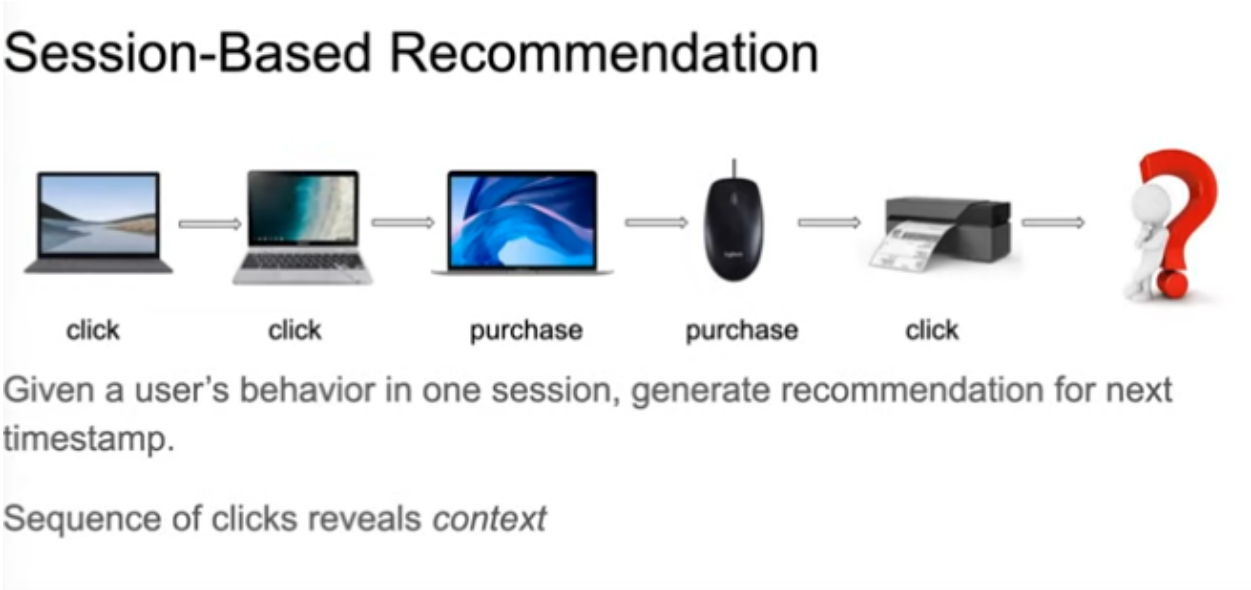
Testing

| Outbrain | k=1 | k=10 | k=20 | k=50 | k=100 |
|---|---|---|---|---|---|
| LinUCB | 0.856 | 0.359 | 0.213 | 0.096 | 0.049 |
| DQN | 0.883 | 0.417 | 0.261 | 0.120 | 0.063 |
| DDQN | 0.881 | 0.403 | 0.246 | 0.113 | 0.059 |
| A2C-F | 0.888 | 0.428 | 0.261 | 0.122 | 0.064 |
| A2C-D | 0.869 | 0.381 | 0.240 | 0.109 | 0.057 |
| DADRL | **0.914** | **0.567** | **0.393** | **0.201** | **0.111** |

Table. The Click Through Rate (CTR) of the 6 methods in One Plus K Random Items test setting on the dataset

- One training episodes is a full interaction session with one user (using 5 workers)
- Tested in One-plus-k-random-items setting, and results are the average of 5 runs
- Federated-like setting has more advantage (DADRL, A2C-F)

# Context-Aware Recommender Systems




Model based approaches like recurrent neural network, transformers network, etc here which take a sequence of interactions that the user has already done and try to predict the next interaction. These models are trained by some self supervised training method, that is there is no external label to this data we have only the sequence of events like sequence of clicks for example and we try to predict the next click.

Traditional self-supervised learning methods do not take into the account the following factors.

1. In E-commerce, recommendations should lead to purchases not just clicks.
2. Long term user engagement.
3. Specific requirements like diversity, supply chain constraints, etc. Therefore reinforcement learning comes into picture.

Therefore reinforcement learning comes into picture as it offers flexible reward

setting and long term optimization. The reinforcement learning agent is trained to take action given the state of the environment with the objective of getting the maximum long term rewards. Here the actions are the items we are recommending and and reward is the recommendation system objective like purchase, etc. State is all the previous actions that the user has done upto that point.

## Recommendation System as Markov Decision Process-

- State: The state depends on the current sequence $\mathbf{s}_t = G(x_{1:t}) \in \mathcal{S}(t > 0)$.
- Action: The recommended item $a_t = x_{t+1}$
- $\mathbf{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition probability
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function
- $\rho_0$: initial state
- $\gamma$: discount factor

Q-learning regresses to the Bellman equation

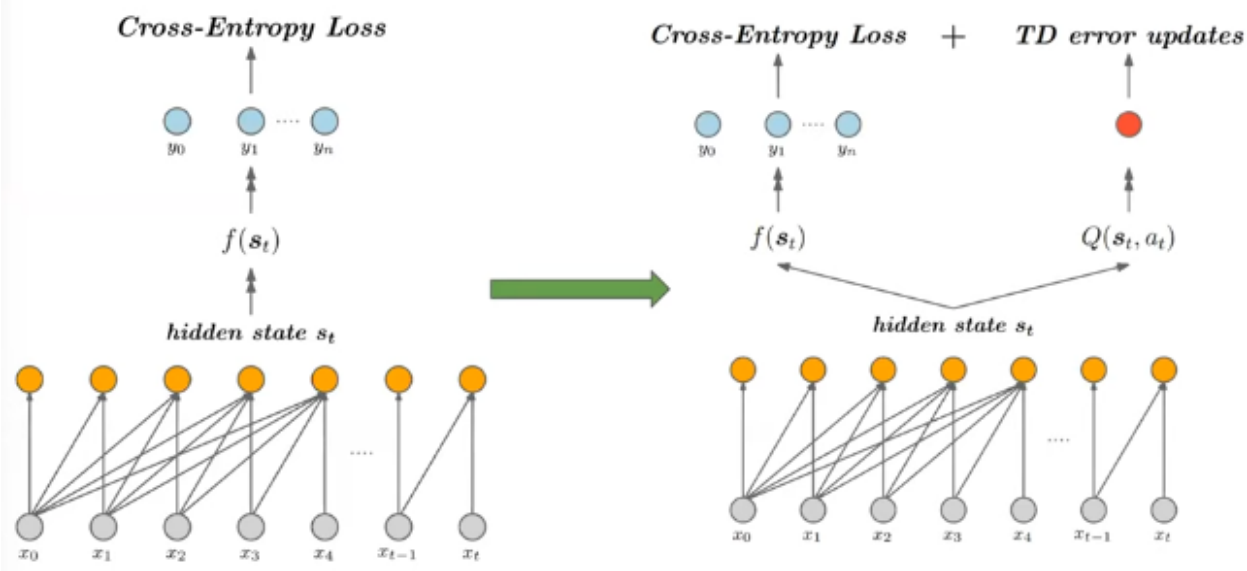
$$Q(\mathbf{s}_t, a_t) = r(\mathbf{s}_t, a_t) + \gamma \max_{a'} Q(\mathbf{s}_{t+1}, a')$$

Actions are selected according to highest Q-values

The off-policy value based approach was used but the issue was that there was no negative feedback. We only know what the users clicked or liked but don't know what the users disliked.

So if we are constantly on positive rewards the recommender gets biased and it recommends anything because the user is going to click on something anyway. Therefore self supervised Q- learning is used. ge

## Self-Supervised Q-learning (SQN)



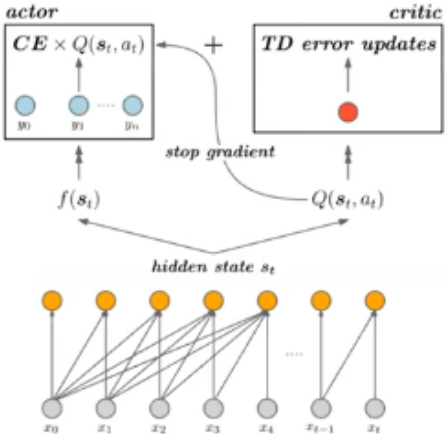

It follows the sequential approach like recurrent neural network, etc. A second output layer is added which is trained with Q-learning which acts a conditioning for the sequential model to perform recommendations that bring close towards the reward which a be a purchase, more diverse recommendation or long term engagement of the user.

## Self-Supervised Actor-Critic (SAC)

- Stop gradient when Q-values are used as weights
- Double Q-learning for stability
- Pretrain with SQN and then train SAC

$$L_{SAC} = L_s \times Q(s, a) + L_q$$

**Both SQN and SAC can be used for most generative recommendation models**



# Experimental Results

Baseline Models:

- GRU [Balázs Hidasi et al, 2015, ICLR]
- Caser [Jiaxi Tang et al, 2018, WSDM]
- NextItNet [Fajie Yuan et al, 2018, WSDM]
- SASRec [Kang et al, 2018, ICDM]

# Datasets and Evaluation metric

Datasets:

- RecSys Challenge 2015 (RC15)
- RetailRocket

| Dataset | RC15 | RetailRocket |
|---|---|---|
| #sequences | 200,000 | 195,523 |
| #items | 26,702 | 70,852 |
| #clicks | 1,110,965 | 1,176,680 |
| #purchase | 43,946 | 57,269 |

Reward Settings:

- Purchase vs Click

Evaluation: (Top-K Recommendation)

- Hit Ratio@K (HR@K)
- NDCG@K

$$\text{HR(click)} = \frac{\#\text{hits among clicks}}{\#\text{clicks}}$$

# Results (RC15)

| | purchase | | click | |
|---|---|---|---|---|
| | HR@5 | NDCG@5 | HR@5 | NDCG@5 |
| GRU | 0.3994 | 0.2824 | 0.2876 | 0.1982 |
| GRU-SQN | 0.4228 | 0.3016 | **0.3020** | **0.2093** |
| GRU-SAC | **0.4394** | **0.3154** | 0.2863 | 0.1985 |
| | | | | |
| Caser | 0.4475 | 0.3211 | 0.2728 | 0.1896 |
| Caser-SQN | 0.4553 | 0.3302 | **0.2742** | **0.1909** |
| Caser-SAC | **0.4866** | **0.3527** | 0.2726 | 0.1894 |

| | purchase | | click | |
|---|---|---|---|---|
| | HR@5 | NDCG@5 | HR@5 | NDCG@5 |
| NextItNet | 0.3632 | 0.2547 | 0.2950 | 0.2030 |
| NextItNet-SQN | 0.3845 | 0.2736 | **0.3091** | **0.2137** |
| NextItNet-SAC | **0.3914** | **0.2813** | 0.2977 | 0.2055 |
| | | | | |
| SASRec | 0.4228 | 0.2938 | 0.3187 | 0.2200 |
| SASRec-SQN | 0.4336 | 0.3067 | **0.3272** | **0.2263** |
| SASRec-SAC | **0.4540** | **0.3246** | 0.3130 | 0.2161 |

## Conclusion

1. Both SQN and SAC archives better performance when predicting purchase behaviour
2. SQN archives best performance for click prediction.
3. SAC archives best performance for purchase prediction.

# Purchase Intent and Budget Prediction in session data

Dynamically detecting the purchase intent and predicting the final purchased price of an item in an ongoing session will help in dynamically adjusting the advertisement strategies and personalized promotions Thus it will help improve the recommendation quality.

## Modeling Actions, Contexts, and Content

- **Action Type**: view, click, add-to-cart, add-to-wishlist, purchase…
- **Content:** price, category, …
- **Context:** timespan (from the last action to the current), device, location, repeat count, …

- We embed all the above into a *d*-dimensional space
- **Discrete features:** one-hot embeddings
- **Continuous features:** feature quantization (bucketization)

- **Count embeddings:** user may repeatedly access the same product, we record the count as a feature

## Embedding Aggregation

- We embed each action *a* in a session (*a1*, *a2*, *a3*, …)
- Action emb **=** Item emb **+** (pos emb) **+** all type/content/context embs

- We also tried ***concatenation***, but didn't observe better performance (presumably due to the high dimensionality)

- **Task:** at each time step *t*, given embeddings of actions *(a1, …, at)*, predict budgets and purchase intent

## Models used for prediction

- **Bag of Items (BOI)** A simple method that uses the summation of embeddings of previous actions (i.e., a1~at). This approach ignores the order of actions

- **Recurrent Neural Networks (RNN)** A classic neural network for handling sequence data. We feed action embeddings into a LSTM

- **Temporal Convolutional Networks (TCN)** A recently proposed method based on 1D dilated convolutions, achieves promising results on various NLP tasks compared with RNNs

- **Self-Attention** a dynamic prediction model, similar to SASRec

# Datasets

| | YooChoose | Internal |
|---|---|---|
| #Sessions | 530,812 | 954,141 |
| #Items | 31,297 | 280,928 |
| #Total actions | 3,450,054 | 6,390,346 |
| #Avg. session length | 6.5 | 6.7 |
| #Categories | 205 | 1656 |
| Content & Context | category, timestamp | action type, price, category, timestamp |
| Time Span | 2014/3/31-2014/9/29 | 2018/11/7- 2018/12/2 |
| #Repeated clicks in *purchase sessions* | 45.6% | 26.7% |
| #Repeated clicks in *non-purchase sessions* | 20.3% | 3.9% |

Table 2. Data Statistics (after preprocessing). A repeated click means the clicked item has been viewed before.

# Performance of intent prediction (AUC)

| | YooChoose | | | | Internal | | | |
|---|---|---|---|---|---|---|---|---|
| | BOI | RNN | TCN | Self-Attention | BOI | RNN | TCN | Self-Attention |
| **item ID** | 0.6073 | 0.6234 | 0.6217 | 0.6582 | 0.6335 | 0.6444 | 0.6476 | 0.6343 |
| **item ID+Content+Context** | 0.6103 | 0.6352 | 0.6371 | 0.6619 | 0.6395 | 0.6629 | 0.6622 | 0.6465 |
| **item ID+Content+Context+Count** | 0.6171 | 0.6444 | 0.6443 | **0.6696** | 0.6444 | **0.6708** | 0.6583 | 0.6509 |

Table 3. AUC on dynamic purchase intent prediction.

# Performance of Budget prediction (Accuracy)

| | YooChoose | | Internal | |
|---|---|---|---|---|
| | Top-1 | Top-5 | Top-1 | Top-5 |
| Popular Price | 0.0801 | 0.2739 | 0.1009 | 0.3334 |
| Avg. Price of Clicked Items | - | - | 0.1539 | 0.4730 |
| **item ID** | | | | |
| BOI | 0.3600 | 0.6945 | 0.1948 | 0.5005 |
| RNN | 0.3704 | 0.6999 | 0.2073 | 0.5213 |
| TCN | 0.3525 | 0.6731 | 0.2375 | 0.5389 |
| Self-Attention | 0.3730 | 0.7114 | 0.2306 | 0.5366 |
| **item ID+Content+Context** | | | | |
| BOI | 0.3678 | 0.7018 | 0.2641 | 0.5989 |
| RNN | 0.3752 | 0.7111 | 0.3352 | 0.6254 |
| TCN | 0.3542 | 0.6801 | 0.3433 | 0.6236 |
| Self-Attention | **0.3765** | **0.7152** | **0.3441** | **0.6292** |

Table 4. Accuracy on dynamic budget prediction.

# References


1. https://www.youtube.com/watch?v=3VjLWN6Nulw
2. https://www.youtube.com/watch?v=qAr9YJlMqGk
3. https://www.youtube.com/watch?v=KoMKgNeUX4k
4. https://www.baeldung.com/cs/ml-policy-reinforcement-learning
5. https://towardsdatascience.com/model-based-reinforcement-learning-cb9e41ff1f0d