Selected ICLR 2017 Papers

Agnieszka Pocha

Jagiellonian University

19.05.2017, Kraków

Outline

Designing Neural Network Architectures using Reinforcement Learning

Designing Neural Network Architectures using Reinforcement Learning

Bowen Baker, Otkrist Gupta, Nikhil Naik, Ramesh Raskar

Abstract

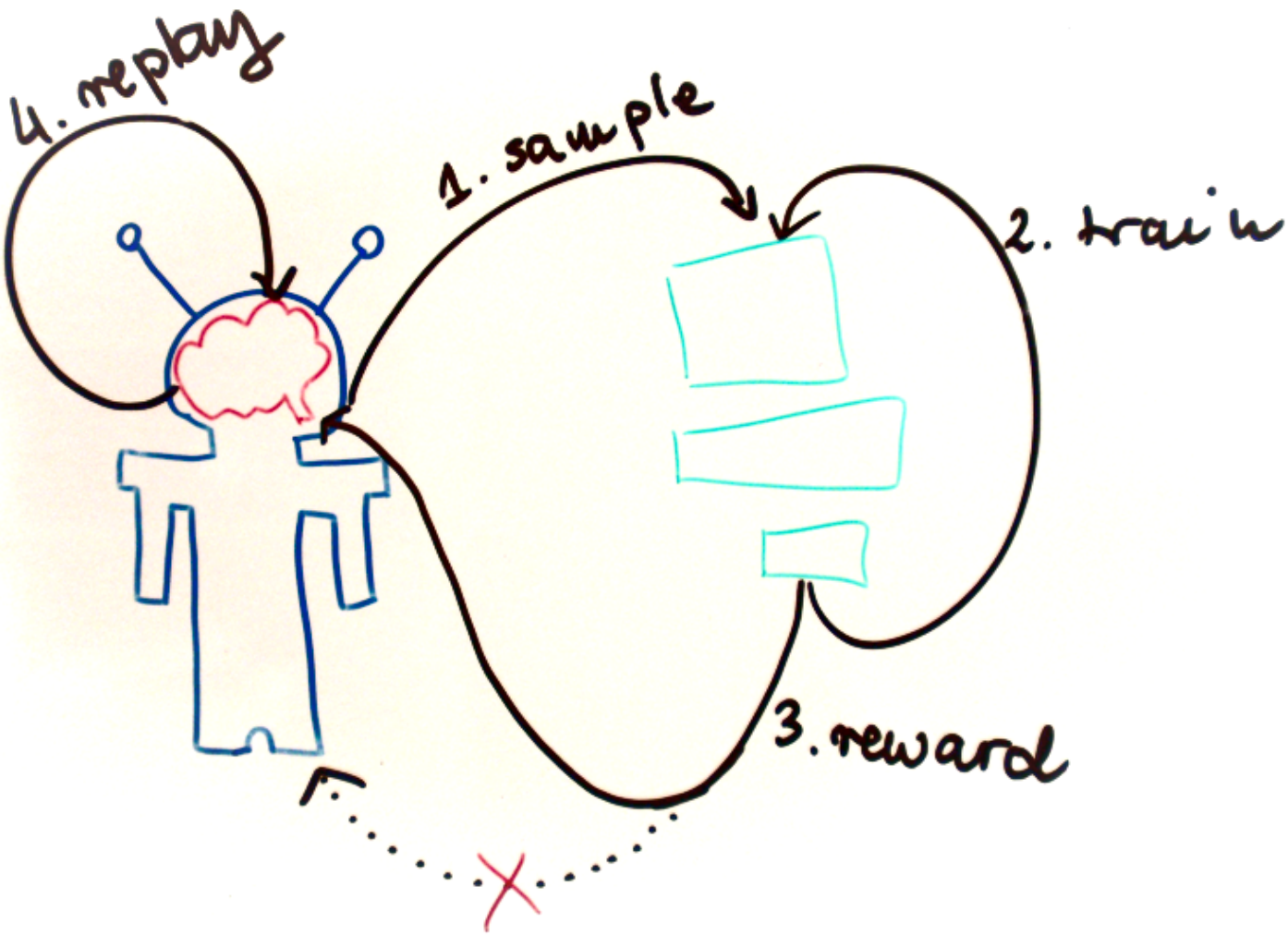
Designing NN architectures is slow and laborious. We would like to have an automatic and successful method to do it for us. Authors introduce MetaQNN - a **reinforcement learning method** based on **Q-learnig** algorithm that finds highly performing architectures.

Their models:

- beat on CIFAR, MNIST & SVHN best performing models that share similar architecture
- compare well with state-of-the-art models
- are suitable for transfer learning

Moreover, authors show that their method is stable.

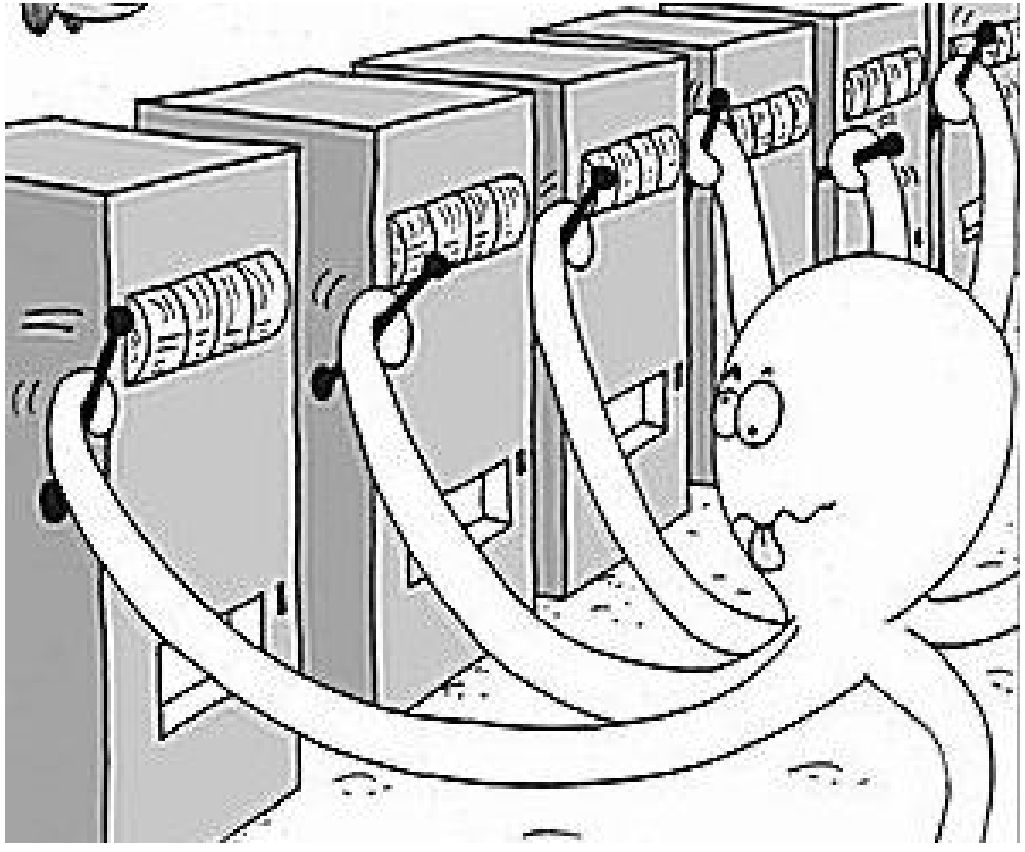
Shortly



Learning agent sequentially chooses CNN layers using **Q-learnig** with an ϵ-**greedy exploration strategy** and **experience replay**.

ϵ-greedy exploration strategy

Choose what you think is the best option with probability $1 - \epsilon$ and choose a random action with probability ϵ.



http://research.microsoft.com/en-us/projects/bandits/

Q-learnig

Let $\mathcal{S}$ - state space, $\mathcal{U}$- action space, $\mathcal{U}(s_i) \in \mathcal{U}$ - actions possible while in state s_i.

In an environment with stochastic transitions, an agent in state s_i taking some action $u \in \mathcal{U}$ will **transition** to state s_j with probability $p_{s'|s,u}(s_j|s_i, u)$ which may be unknown to the agent.

At each timestep t, the agent is given a **reward** r_t, dependent on the transition from state s to s' and action u. The reward may also be stochastic according to a distribution $p_{r|s',s,u}$.

The agent's **goal** is to maximize the **total expected reward** over all possible trajectories, i.e. $max_{\tau_i \in \tau} R_{\tau_i}$, where the total expected reward for a trajectory τ_i is

$$R_{\tau_i} = \Sigma_{(s,u,s') \in \tau_i} \mathbb{E}_{r|s,u,s'}[r|s, u, s']$$

Q-learnig

Let $\mathcal{S}$ - state space, $\mathcal{U}$- action space, $\mathcal{U}(s_i) \in \mathcal{U}$ - actions possible while in state s.

$$R_{\tau_i} = \Sigma_{(s,u,s') \in \tau_i} \mathbb{E}_{r|s,u,s'}[r|s,u,s']$$

The number of possible trajectories makes the problem untractable, therefore we define the maximization problem recursively in terms of subproblems as follows: for any state $s_i \in S$ and subsequent action $u \in \mathcal{U}(s_i)$, we define the **maximum total expected reward** to be $Q^*(s_i, u)$. The recursive maximization equation, which is known as **Bellman's Equation**, can be written as:

$$Q^*(s_i, u) = \mathbb{E}_{s_j|s_i,u}[\mathbb{E}_{r|s_i,u,s_j}[r|s_i,u,s_j] + \gamma max_{u' \in \mathcal{U}(s_j)} Q^*(s_j, u')]$$

We usually cannot solve it analytically but we can define an **iterative update**.

Q-learning

- **model-free**
- **off-policy**
- two parameters:
  - α - **learning rate**
  - λ - **discount factor** (weight given to short term rewards over future rewards)

Here:

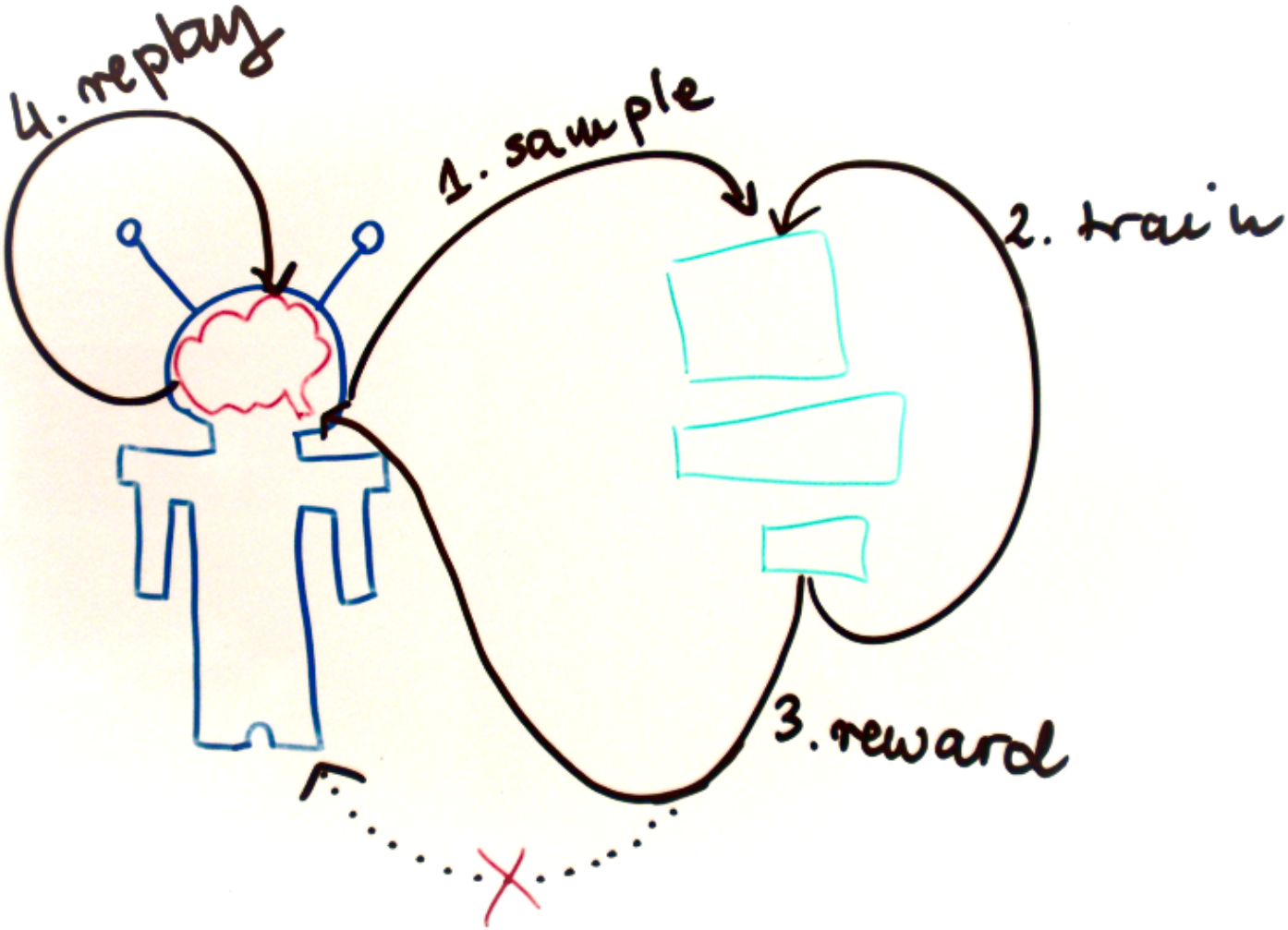
- **action** - choosing next layer along with its parameters (size, stride...)
- **state** - the current state of the network topology
- **reward** - performance on validation set (5K samples, with unchanged class distribution)

Experience replay

Use single experience multiple times for an iterative update.

https://www.shutterstock.com/image-photo/empty-toilet-paper-roll-recycled-seedling-134967662?src=ijY4nrNlx43dUDT2n_Cjsw-1-4

Shortly (again)



Learning agent sequentially chooses CNN layers using **Q-learnig** with an ϵ-**greedy exploration strategy** and **experience replay**.

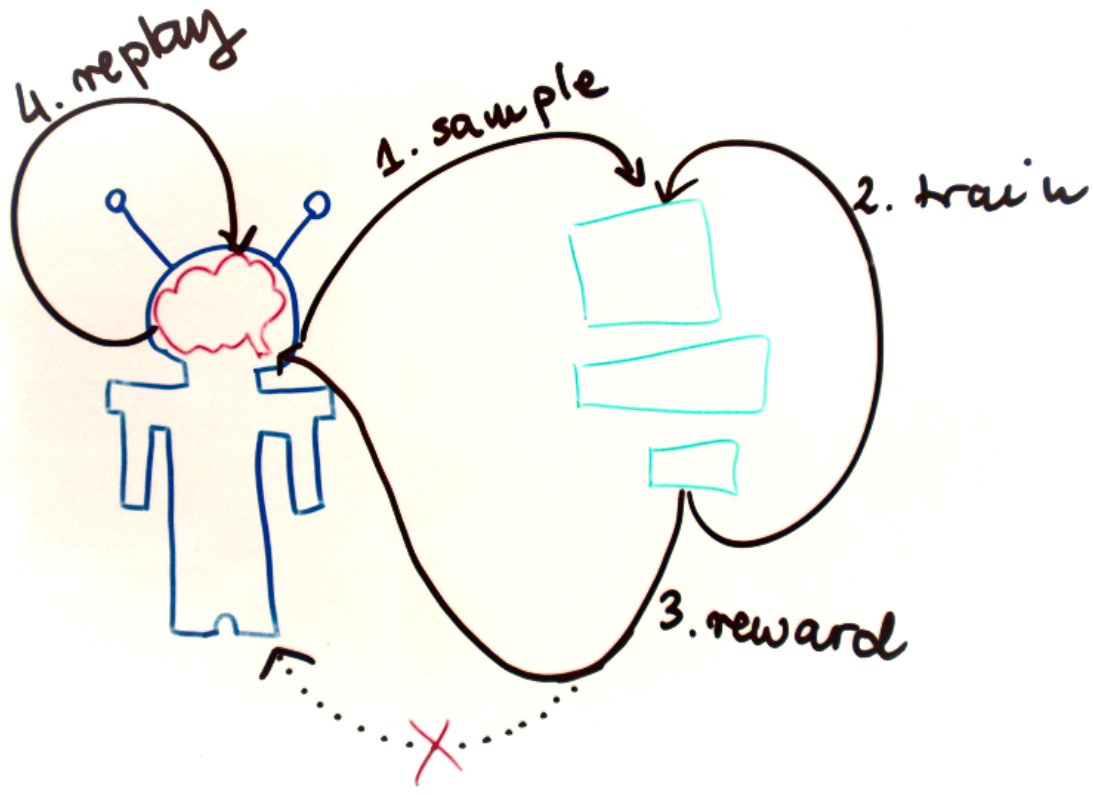
Training models

Models were trained in an aggresive fashion (stop this violence!) what made the process less costly.

- dropout every 2 layers
- 20 epochs
- Adam
- batch size 128
- learning rate 0.001 reduced by a factor of 0.2 every 5 epochs
- Xavier initialization (Glorot & Bengio 2010)

Time: 8-10 days for each dataset on 10 NVIDIA GPUs (GMUM needs more GPUs).

Again same picture but now we should understand it completely



Learning agent sequentially chooses CNN layers using **Q-learnig** with an ϵ-**greedy exploration strategy** and **experience replay**.

Experiments

It grows! It means that the architectures chosen greedily/later (exploitation) are better than those randomly sampled (exploration) - the agent has learned something.

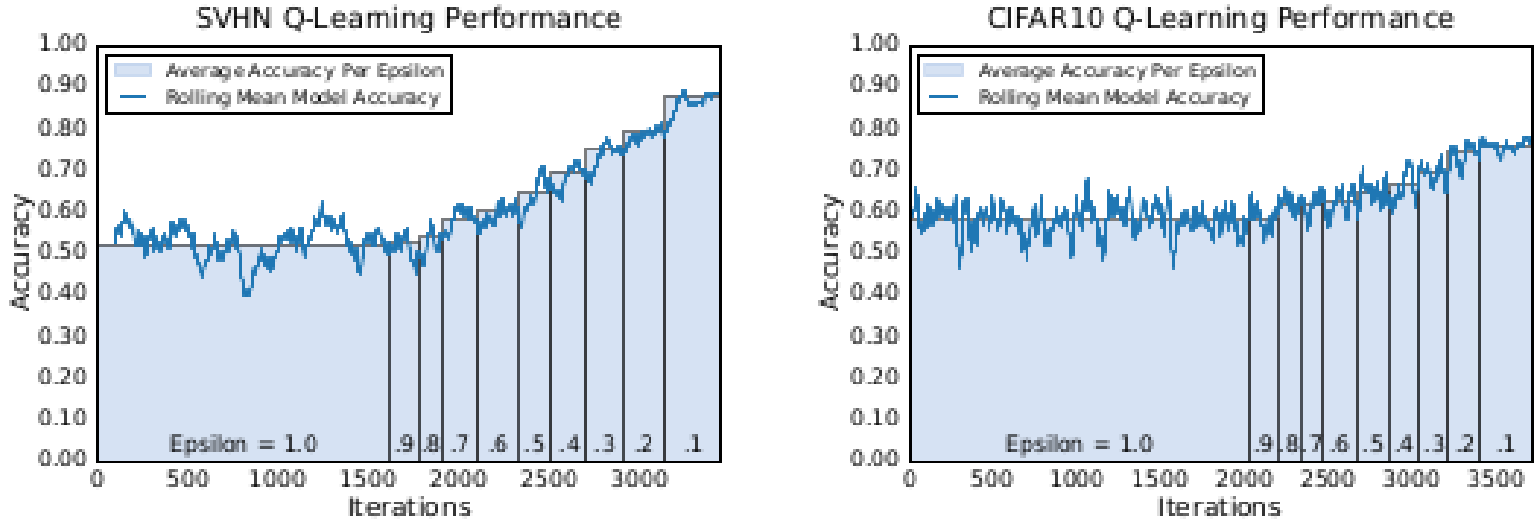
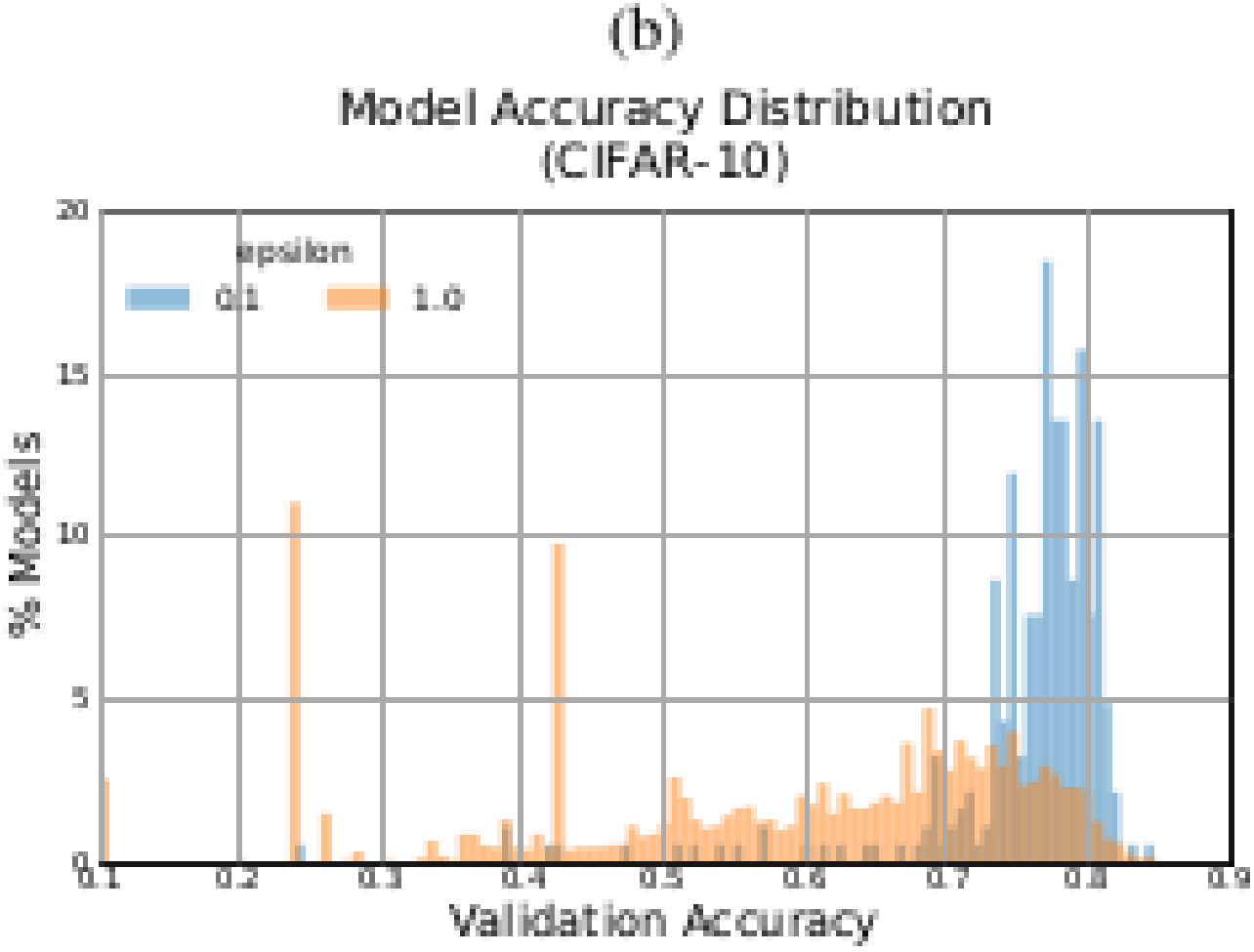


Figure 3: ***Q*-Learning Performance.** In the plots, the blue line shows a rolling mean of model accuracy versus iteration, where in each iteration of the algorithm the agent is sampling a model. Each bar (in light blue) marks the average accuracy over all models that were sampled during the exploration phase with the labeled ϵ. As ϵ decreases, the average accuracy goes up, demonstrating that the agent learns to select better-performing CNN architectures.

Experiments

It learns! It means that the architectures chosen greedily/later (exploitation) are better than those randomly sampled (exploration) - the agent has learned something.



Results

After short training all those architectures 10 best were chosen and well tuned. Finally, 5 best were chosen to become ensemble.

- beat on CIFAR, MNIST & SVHN best performing models that share similar architecture

Method	CIFAR-10	SVHN	MNIST	CIFAR-100
Maxout (Goodfellow et al., 2013)	9.38	2.47	0.45	38.57
NIN (Lin et al., 2013)	8.81	2.35	0.47	35.68
FitNet (Romero et al., 2014)	8.39	2.42	0.51	35.04
HighWay (Srivastava et al., 2015)	7.72	-	-	-
VGGnet (Simonyan & Zisserman, 2014)	7.25	-	-	-
All-CNN (Springenberg et al., 2014)	7.25	-	-	33.71
MetaQNN (ensemble)	7.32	**2.06**	**0.32**	-
MetaQNN (top model)	**6.92**	2.28	0.44	**27.14***

Results

- compare well with state-of-the-art models

Method	CIFAR-10	SVHN	MNIST	CIFAR-100
DropConnect (Wan et al., 2013)	9.32	1.94	0.57	-
DSN (Lee et al., 2015)	8.22	1.92	0.39	34.57
R-CNN (Liang & Hu, 2015)	7.72	1.77	**0.31**	31.75
MetaQNN (ensemble)	7.32	2.06	0.32	-
MetaQNN (top model)	6.92	2.28	0.44	27.14*
Resnet(110) (He et al., 2015)	6.61	-	-	-
Resnet(1001) (He et al., 2016)	**4.62**	-	-	**22.71**
ELU (Clevert et al., 2015)	6.55	-	-	24.28
Tree+Max-Avg (Lee et al., 2016)	6.05	**1.69**	**0.31**	32.37

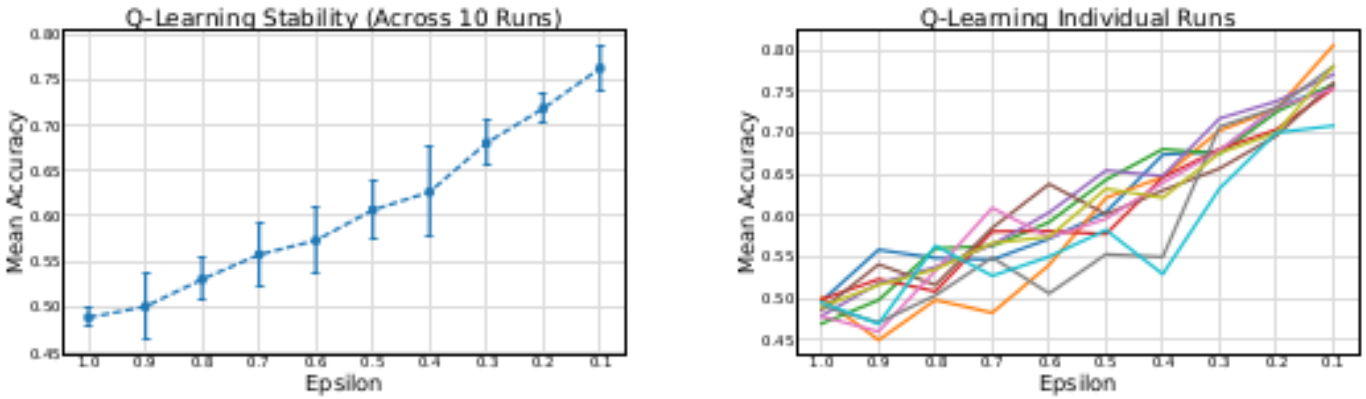
- are suitable for transfer learning

Dataset	CIFAR-100	SVHN	MNIST
Training from scratch	27.14	2.48	0.80
Finetuning	34.93	4.00	0.81
State-of-the-art	24.28 (Clevert et al., 2015)	1.69 (Lee et al., 2016)	0.31 (Lee et al., 2016)

Table 5: **Prediction Error** for the top MetaQNN (CIFAR-10) model trained for other tasks. Fine-tuning refers to initializing training with the weights found for the optimal CIFAR-10 model.

Results

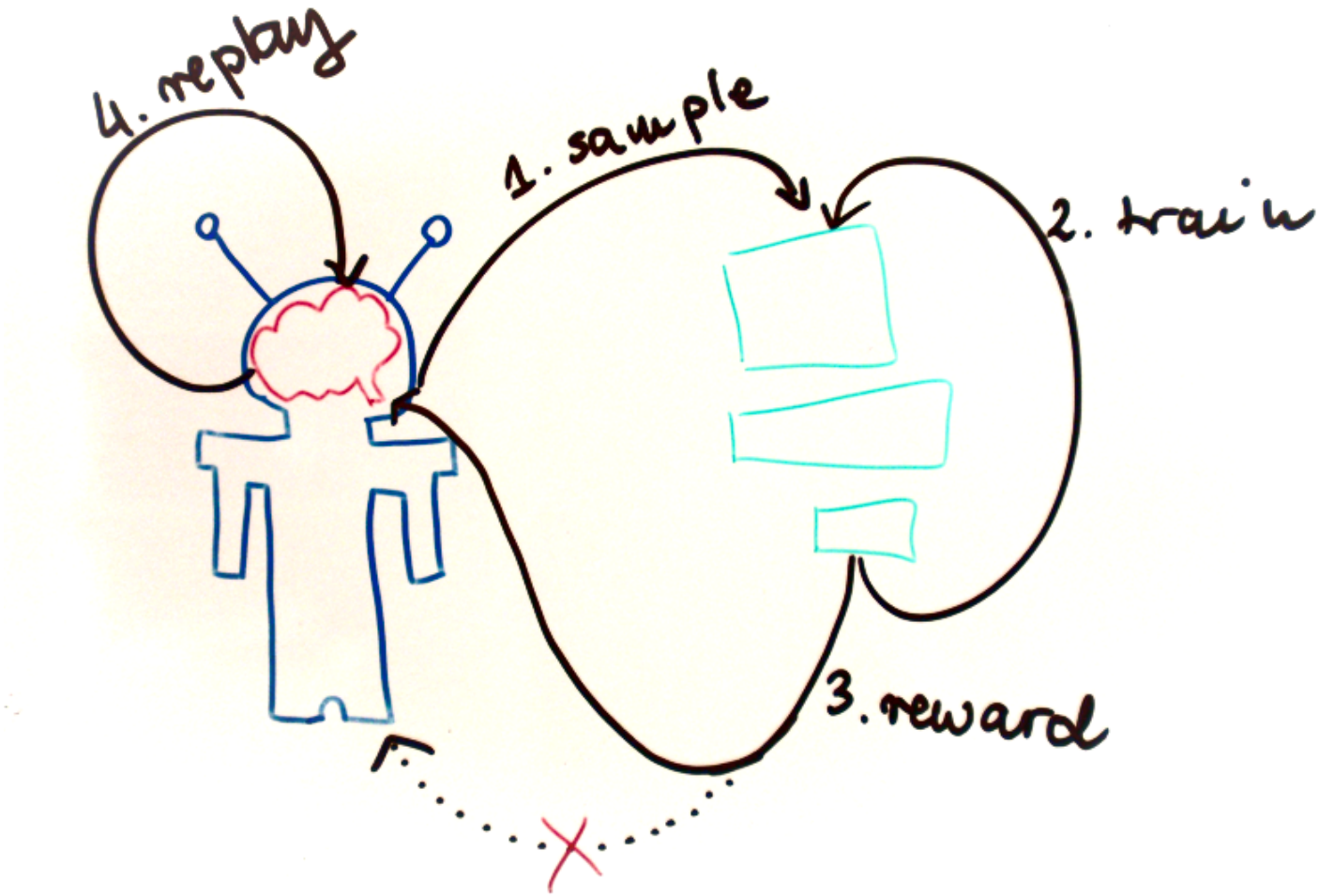
- stability



(a) (b)

Figure A3: Figure A3a shows the mean model accuracy and standard deviation at each ϵ over 10 independent runs of the Q-learning procedure on 10% of the SVHN dataset. Figure A3b shows the mean model accuracy at each ϵ for each independent experiment. Despite some variance due to a randomized exploration strategy, each independent run successfully improves architecture performance.

My problem with this paper



Late architectures (best performing) might not be chosen for replay i.e. not used for training at all while early architectures (randomly performing) will be overrepresented.

My problem with this paper

Algorithm 1 Q-learning For CNN Topologies

Initialize:
replay_memory $\leftarrow [\,]$
$Q \leftarrow \{(s, u)\ \forall s \in \mathcal{S}, u \in \mathcal{U}(s)\ :\ 0.5\}$
for episode = 1 to M **do**
$S, U \leftarrow$ SAMPLE_NEW_NETWORK(ϵ, Q)
accuracy $\leftarrow$ TRAIN(S)
replay_memory.append((S, U, accuracy))
for memory = 1 to K **do**
S_{SAMPLE}, U_{SAMPLE}, accuracy$_{SAMPLE}$ $\leftarrow$ Uniform{replay_memory}
$Q \leftarrow$ UPDATE_Q_VALUES(Q, S_{SAMPLE}, U_{SAMPLE}, accuracy$_{SAMPLE}$)
end for
end for

Late architectures (best performing) might not be chosen for replay i.e. not used for training at all, while early architectures (randomly performing) will be overrepresented.

WHY?

Important note

"While we report results for image classification problems, our method could be applied to different problem settings, including supervised (e.g., classification, regression) and unsupervised (e.g., autoencoders)"

More

- arXiv 1611.02167 `https://arxiv.org/abs/1611.02167`
- GitHub `https://bowenbaker.github.io/metaqnn/`