Designing Neural Network Architectures Using Reinforcement Learning

Presented by: Andrew Low

CS 294 | 2/23/2019

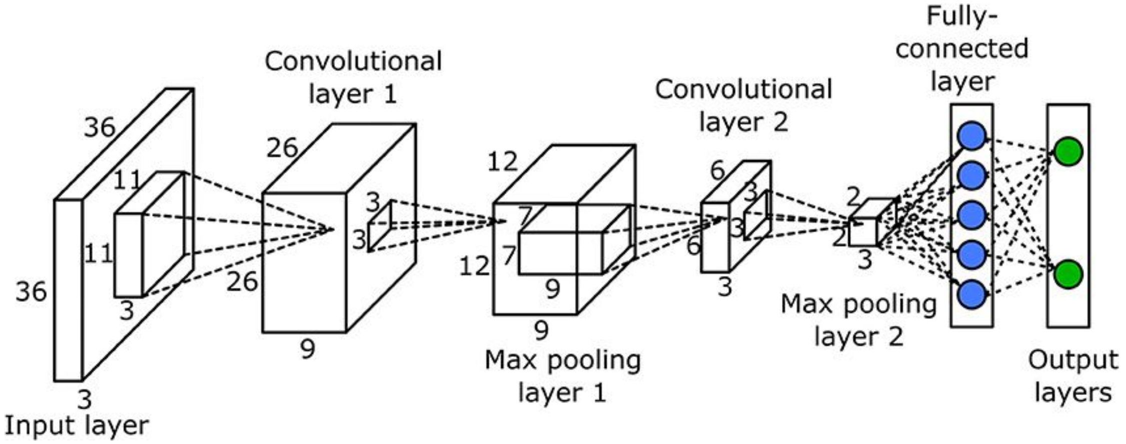
Outline

Context

Neural Networks are powerful and increasingly popular

Many different network architectures exist - without a clear winner

Architecture depends on the domain



Problem

Convolutional neural network architecture design today

- Large search space
- Most novel architectures are hand-designed, motivated by theoretical insights and experimental intuition of experts
- Slow and expensive!

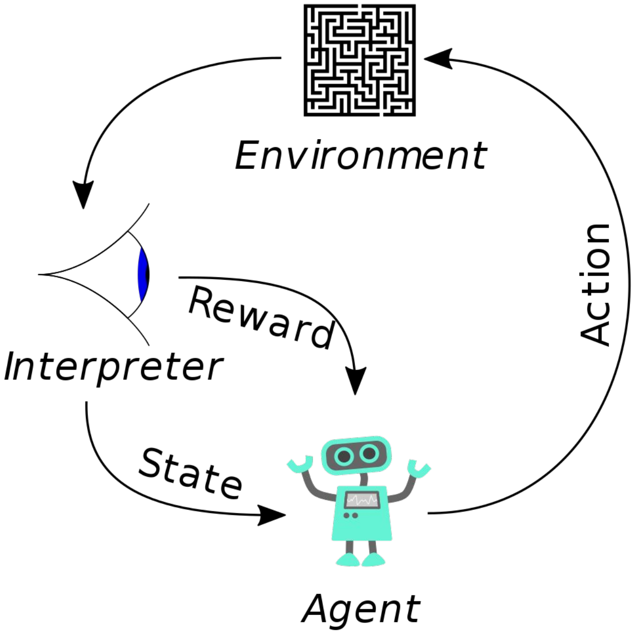
<u>How to efficiently find optimal neural net architectures?</u>

Background - Reinforcement Learning Recap

State space S, action space U, and reward distribution R.

Rewards may be delayed and/or sparse - require a sequence of correct actions

Goal: Find the optimal policy that maximizes our expected reward (Find optimal path on a MDP with a finite horizon)

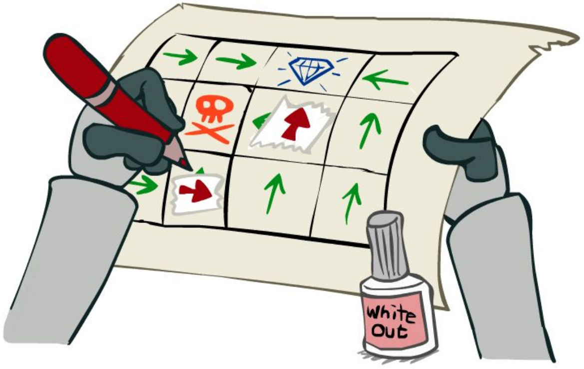


Background - Q Learning

Difficult to know the actual value function, so we approximate the value function using Q values

Model free and Off-policy

As the agent explores the state and action spaces, it learns about its environment and retains that knowledge via Q values



Background - Exploration and Exploitation

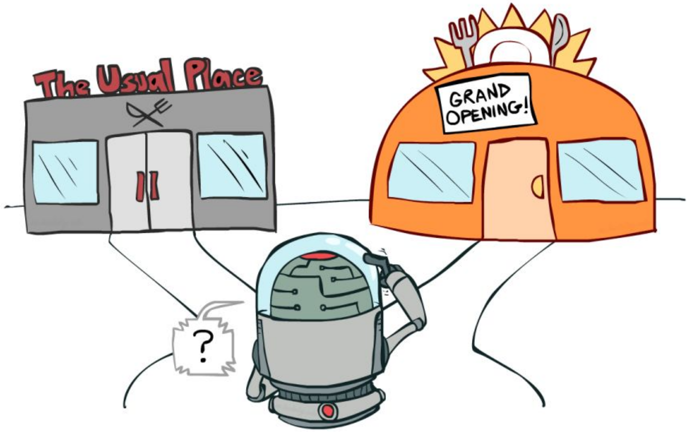
Exploration: when an agent tries new actions and states to learn about its environment

Exploitation: when an agent utilizes what it knows to take the best path possible

Too much exploration -> slow convergence

Too much exploitation -> converge to local optima

ε-learning: Higher ε means more exploration

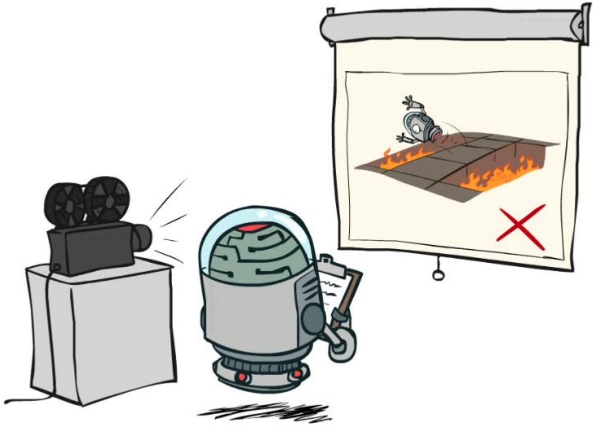


Background - Experience Replay

Generating data for reinforcement learning can be costly - and many RL algorithms require lots of data

We store each (state, action, reward, new state) in a database

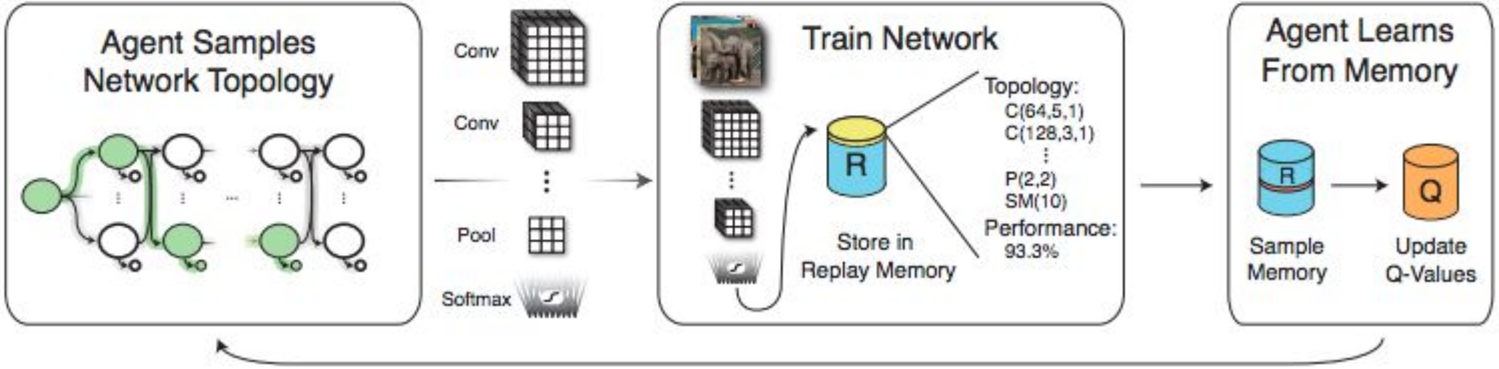
Can then ‘replay’ past experiences by randomly sampling from the database



Reformulating the Problem

<u>The key innovation is to reformulate the network architecture search as a reinforcement learning task!</u>

- State space: all possible neural net architectures
- Action space: choosing new layers (conv, FC, pool) to put in the network
- Reward function: the validation accuracy of the complete model

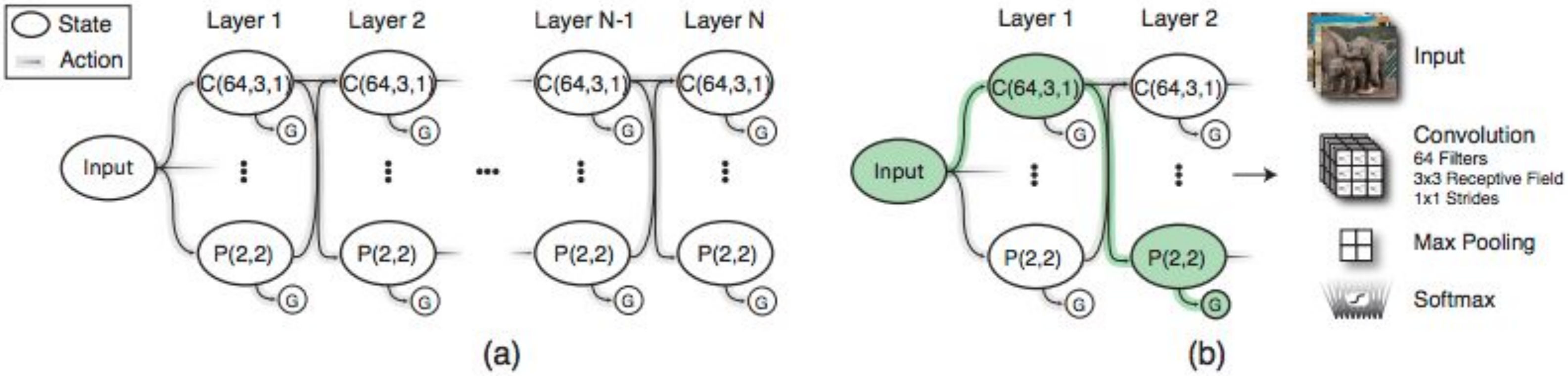


Reformulating the Problem

Key Assumption - a well-performing layer in one network will also perform well in a different network

State space - Neural net architectures that can be built using the following layer types: Convolution, Pooling, Fully Connected, Global Average Pooling, and ReLU

Termination states are GAP and Softmax

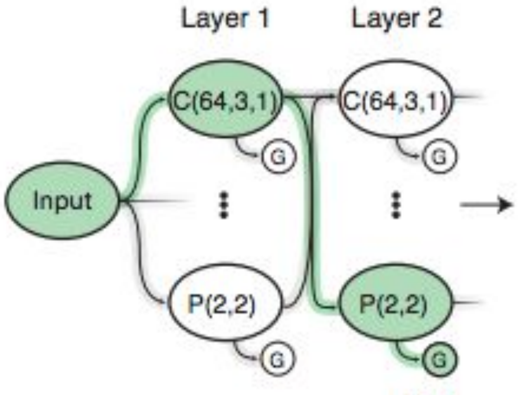


Reformulating the Problem

Action Space - the set of possible layers we can put at the next level.

The authors place restrictions on the action space for tractability

- Maximum network depth
- Representation size
- Layer order
    - Consecutive Pooling layers
    - Transitioning to FC layers
- Number of FC layers



(b)

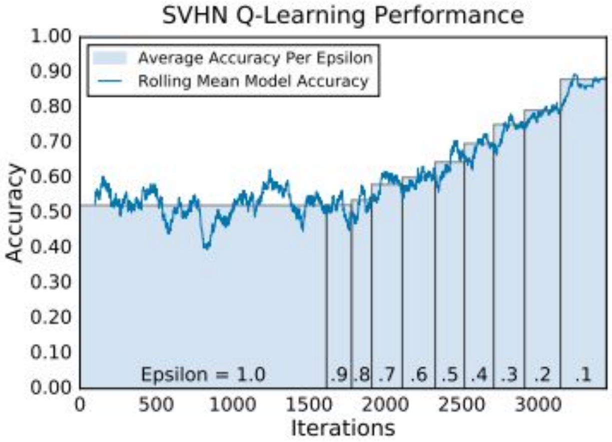
Experimental Setup

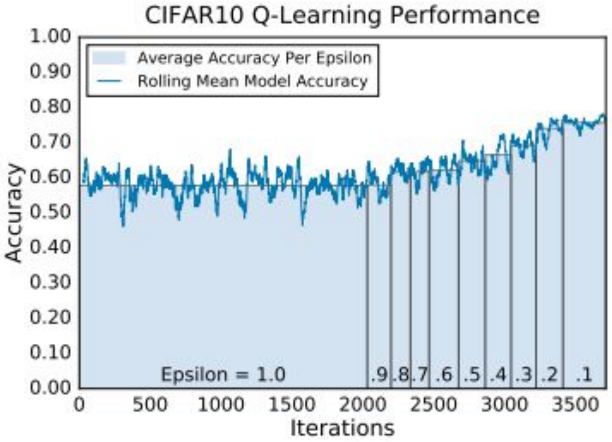
Models were trained with the Adam optimizer

Top ten models were selected and fine tuned further

3 Datasets:

- MNIST
- CIFAR 10
- SVHN





ϵ	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
# Models Trained	1500	100	100	100	150	150	150	150	150	150

Experimental Setup - Details

Each model trained with Adam optimizer

Q-learning rate alpha = 0.01

Epsilon transitions from 1 -> 0.1

Utilizes experience replay to save time

$\beta 1 = 0.9$, $\beta 2 = 0.999$, $\varepsilon = 10-8$

Batch size: 128, Learning rate = 0.001

Key Results

MetaQNN models outperformed CNNs that only used the same layer types

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***

Table 3: **Error Rate Comparison** with CNNs that only use convolution, pooling, and fully connected layers. We report results for CIFAR-10 and CIFAR-100 with moderate data augmentation and results for MNIST and SVHN without any data augmentation.

Key Results

MetaQNN models performed worse than but still at a ‘competitive’ level compared to than state-of-the-art models that utilize more complex layers and training methods.

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

Table 4: **Error Rate Comparison** with state-of-the-art methods with complex layer types. We report results for CIFAR-10 and CIFAR-100 with moderate data augmentation and results for MNIST and SVHN without any data augmentation.

Key Results

MetaQNN models outperformed other automated network design protocols

	CIFAR-10	MNIST
MetaQNN	**6.92**	**0.32**
Bergstra	21.2	
Verbancsics		7.9

Error rates (%)

Limitations and Improvements

Limitations

- Hyperparameter optimization
- Is CNN architecture the limiting factor in model accuracy? Or simply an optimization?

Improvements

- Complex layer types
- More fine-grained state-action space

Impact and Discussion

MetaQNN provides an automated solution for CNN architecture

- Saves research time while pinpointing more optimal solutions
- Largely an optimization - future progress will likely come from different areas

Discussion

- How useful is this program today, given that state-of-the-art models all utilize complex layer types and specialized training techniques?
- As it exists, is MetaQNN useful to non-experts?
- Are there any other areas that can be reformulated as RL tasks?
- Would MetaQNN have been able to re-invent recent architecture breakthroughs?