Recitation 4

Monte Carlo

Monte Carlo (MC)

Update Rule:

$$V(S_t) \leftarrow \text{average}(Returns(S_t))$$

Incremental Update:

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

where return is the sum of discounted rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

DP (Value/Policy Iteration):

- Iterate through all possibilities

$$\sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$$

- assumes full knowledge of env

- One step bootstrap: biased estimate
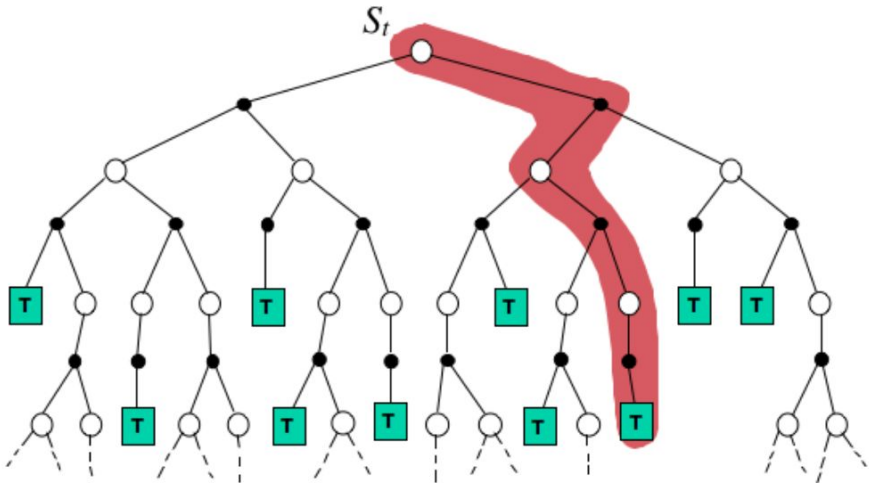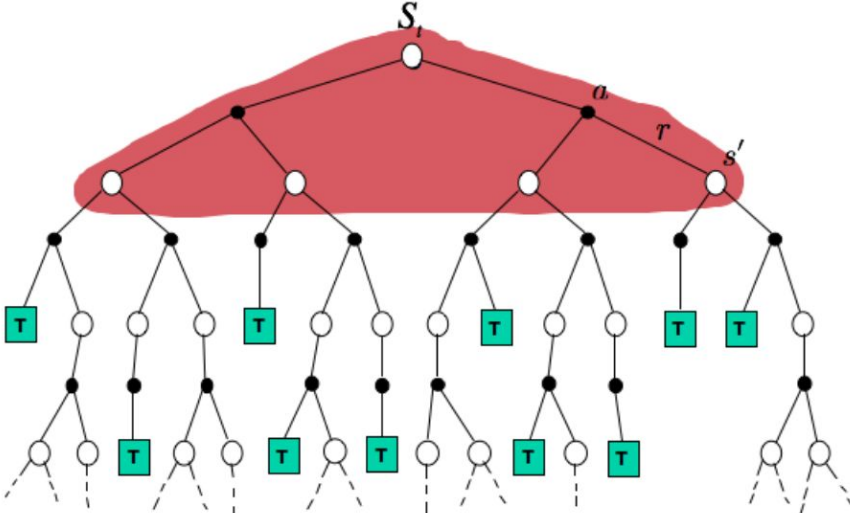
Monte Carlo Learning:

+ Collect samples from episodes

$$\pi: S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$$

What does this assume?

- mean return: unbiased

Pro or con?

DP vs MC

Monte Carlo (MC)

Every-visit also exists… different convergence property

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$\quad V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$\quad Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

$\quad$ Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$\quad G \leftarrow 0$

$\quad$ Loop for each step of episode, $t = T-1, T-2, \ldots, 0$: Aggregate backwards

$\quad\quad G \leftarrow \gamma G + R_{t+1}$ $\qquad\qquad\qquad\qquad$ $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$

$\quad\quad$ Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

$\quad\quad\quad$ Append G to $Returns(S_t)$

$\quad\quad\quad V(S_t) \leftarrow \text{average}(Returns(S_t))$ $\quad$ $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$

How would you modify the
above to also generate a policy?

Temporal Difference Learning

Temporal Difference Learning

New update rule:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

target: an estimate of the return

+ Can learn before reaching a terminal state
+ Much more memory and computation-efficient than MC
- Using value in the target introduces bias

Motivation for TD learning and N-step returns

TD(0)

MC estimate

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

$$= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]$$

$$= \mathbb{E}_\pi \left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s \right]$$

$$= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s \right]$$

Approximate with v

N-step returns

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

$$= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]$$

$$= \mathbb{E}_\pi \left[\sum_{i=1}^{N} \gamma^{i-1} R_{t+i} + \gamma^N \sum_{k=0}^{\infty} \gamma^k R_{t+k+N+1} | S_t = s \right]$$

$$= \mathbb{E}_\pi \left[\sum_{i=1}^{N} \gamma^{i-1} R_{t+i} + \gamma^N v_\pi(S_{t+N}) | S_t = s \right]$$
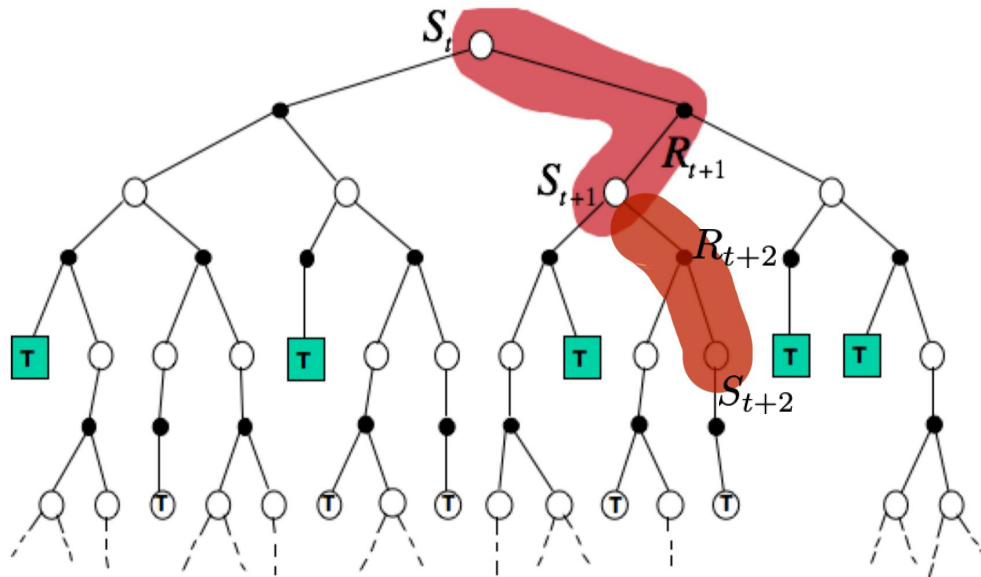
N-step returns   Less reliance on v

N-step returns

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) - V(S_t) \right)$$

Q-learning: Off-policy TD Learning

1-step Q-learning update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

- **Key benefit: off-policy!**
- Only require state, action, reward, and next state drawn from the MDP
- Doesn't depend on the policy anywhere!
- Is foundation for many sample-efficient RL methods

Deep Q-learning

- What happens if the state space and action space are too large?
  - Use function approximation to approximate the Q-values!
- Use gradient descent to take a step towards minimizing the Bellman error:

$$L = \left(\mathrm{sg}(R_{t+1} + \gamma \max_{A_{t+1}} q(S_{t+1}, A_{t+1}, w)) - q(S_t, A_t, w) \right)^2$$

Target value   Prediction

Tabular

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{A_t} q(S_{t+1}, A_{t+1}) - q(S_t, A_t) \right]$$

Function Approximation

$$w \leftarrow w + \alpha \left[R_{t+1} + \gamma \max_{A_t} q(S_{t+1}, A_{t+1}, w) - q(S_t, A_t, w) \right] \nabla_w q(S_t, A_t, w)$$

Target Networks

$$L = \left(\text{sg}(R_{t+1} + \gamma \max_{A_{t+1}} q(S_{t+1}, A_{t+1}, w)) - q(S_t, A_t, w) \right)^2$$

Target value   Prediction

- One problem with deep Q-learning: nonstationary targets
  - Updating the network weights changes the target value, which requires more updates
  - Unintended generalization to other states S' can lead to error propagation
- Solution: calculate target values with a network that's updated every T gradient steps
  - Network has more time to fit targets accurately before they change
  - Slows down training, but not too many alternatives (recently: functional regularization)

Experience Replay

- Problem #1: neural networks undergo **catastrophic forgetting** if they haven't been trained on a (similar) sample recently
- Problem #2: online samples tend to be very correlated, which leads to unstable optimization
- Solution: keep large history of transitions in a "replay buffer," then optimize the Bellman error wrt random minibatches

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

$$I = \left(r + \gamma \max_a Q\left(s', a', \mathbf{w}-\right) - Q(s, a, \mathbf{w}) \right)^2$$

Monte Carlo Tree Search

Problem: Large State-Action Space

Trying to estimate the value at every state (solving the full MDP) is often infeasible

MC and TD still try to estimate Q/V value function for every state or state-action visited
- Too much memory for tabular (10^48 states for chess)
- NN may be undefined at unseen states, "similar" states may have completely different values and optimal paths

Online Planning

- Use internal model to simulate trajectories at current state, find the best one

Monte Carlo Tree Search (MCTS):

- Only estimate value function for relevant part of state space
- Consider only part of the full MDP at a given step

MCTS

node = state

edge = action

Repeat while time remains

Selection — Expansion — Simulation — Backup

Tree Policy

Rollout Policy
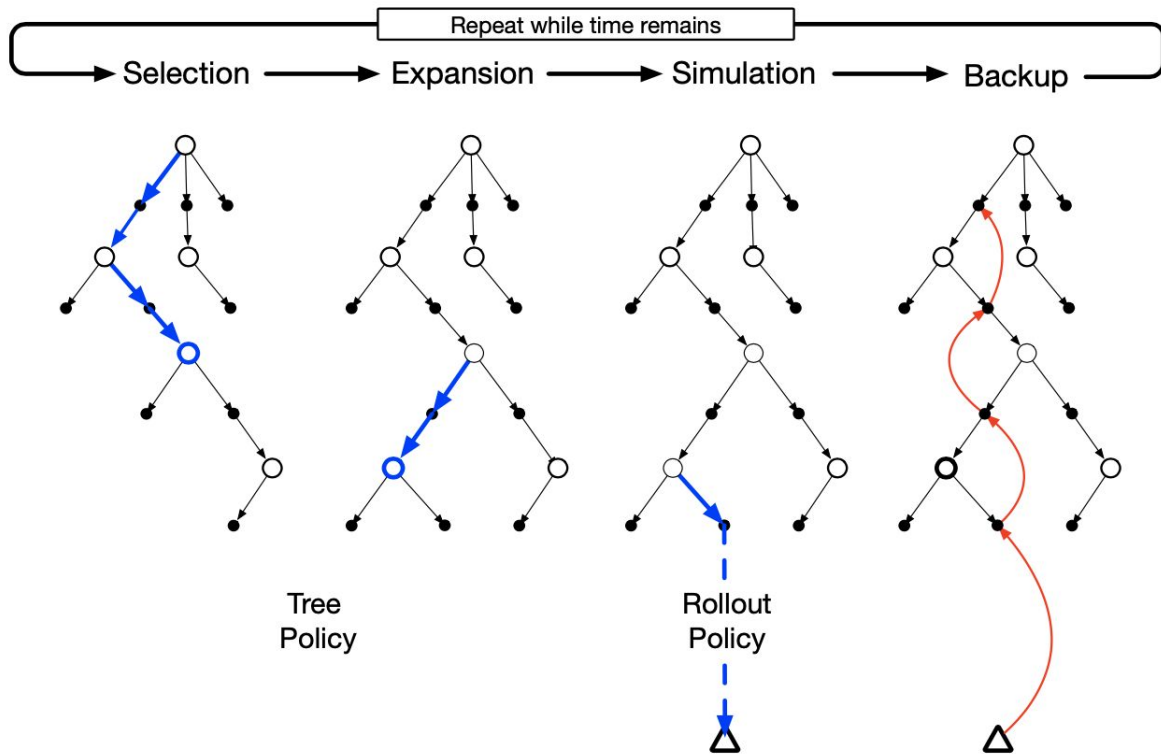
- Tree: Stores Q-values for only a subset of all state-actions
- MC method: require episode termination to update values

Selection

Given:

- current state of agent = root node
- Empty or existing tree with Q-values

Steps:

"children" = actions, don't know all possible (s,a) → s' transitions

```
function MCTS_sample(node)
    if all children expanded: #selection
        next = UCB_sample(node)
        outcome = MCTS_sample(next)
```

$$A_t = \mathrm{argmax}_a \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$

- keep executing UCB repeatedly until you reach frontier of tree (state that is not a node)

Expansion

Given:

-   at a new state **s** not part of the tree

Steps:

Why not store all nodes and Q values?

-   Based on some rule, <u>possibly add this new state to the tree</u>
    -   ex: if depth of this state < max depth
-   Take random action **a** (since no Q-values available), receive reward **r** if available
-   **G** = Simulation(**s**, **a**)
-   Store Q(**s, a**) = gamma***G** + **r**
-   return gamma***G** + **r** to propagate return to parent node
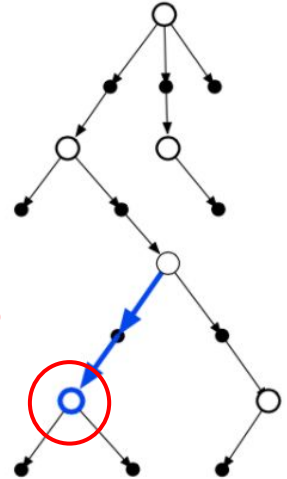
Simulation

Given:

- at a new state **s** not part of the tree

Steps:

- If at terminal state, return reward
- use very fast policy to determine action **a** to take
    - ex: random policy
- **G** = Simulation(**s**, **a**)
- return gamma***G** + **r**   (Do Not store Q-value)

Rollout
Policy
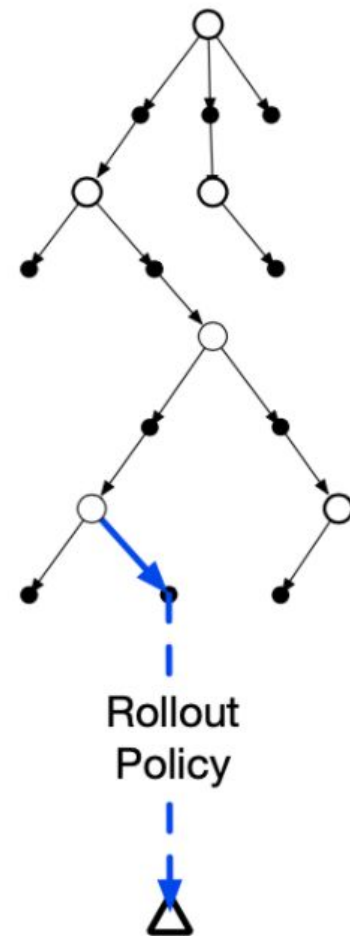
Backup

- Propagate return from the recursive calls
- Calculate return at each state

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$
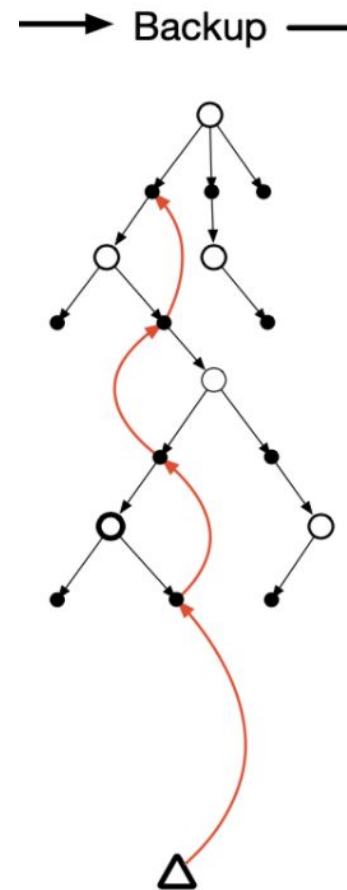
MCTS Overall

- For the current state of agent, repeatedly perform the previous steps until some criteria
    - ex: time limit
    - ex: Q-value convergence within some threshold
- Execute the best action
- Reuse the subtree of the successor state and repeat!

What scenarios would you use MCTS
as opposed to learning?

- available time
- internal model
- size or dynamic nature of
state-action space

Sidenote: ImageNet Dataset

Recitation 4

Fall 2022, CMU 10-703

ImageNet

Dataset for visual object recognition

- > 14 Million images, > 20,000 categories

- Provides ontology over image categories
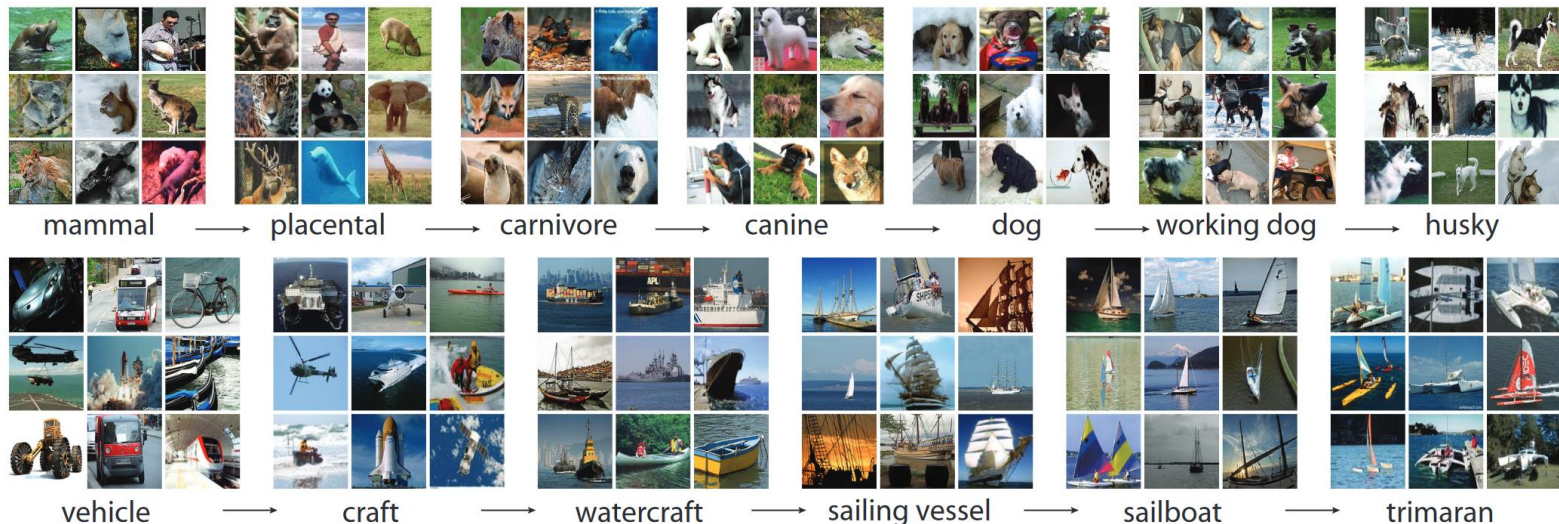
Figure 1: A snapshot of two root-to-leaf branches of ImageNet: the **top** row is from the mammal subtree; the **bottom** row is from the vehicle subtree. For each synset, 9 randomly sampled images are presented.
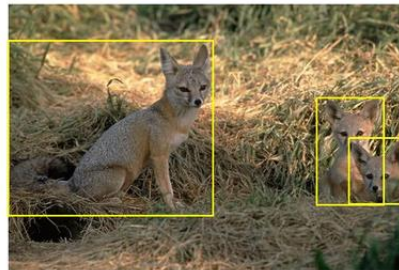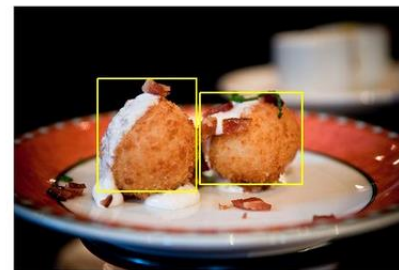
Source: He

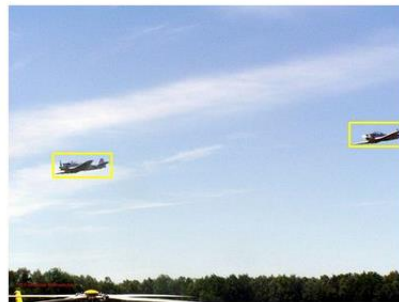2

ImageNet

Dataset for visual object recognition

- Provides bounding boxes for over 3000 classes

- On average 150 images per class
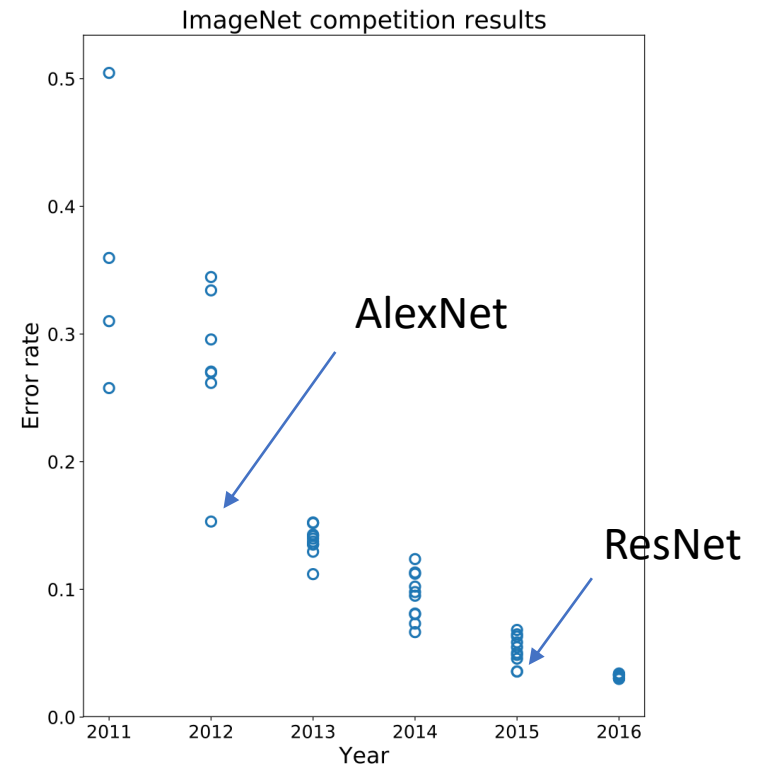

kit fox

croquette

airplane

frog

Source: He

ImageNet

Popular benchmark dataset which has been used for annual competitions and many research papers

ImageNet 2012 Challenge attracted major public attention

-> Deep learning approach reduced top-5 error by 10.8% compared to other machine learning approaches

Start of "deep learning revolution"
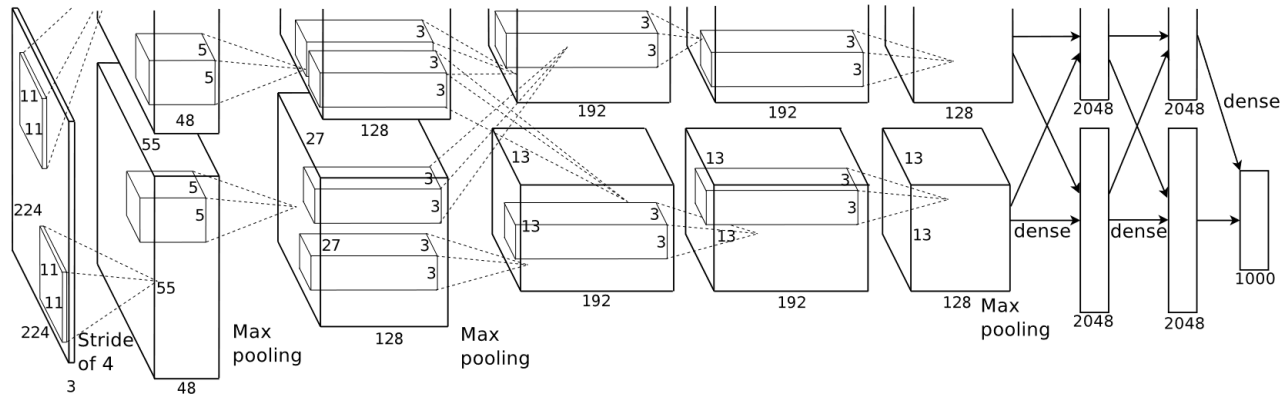

Source: Wikipedia

AlexNet

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Source: Krizhevsky

- Architectures features two lines of CNN networks which are deployed on separate GPUs

- Winner of ImageNet 2012 Challenge (top-5 error of 15.3%, 10.8% lower than 2nd best)

Residual Network (ResNet)
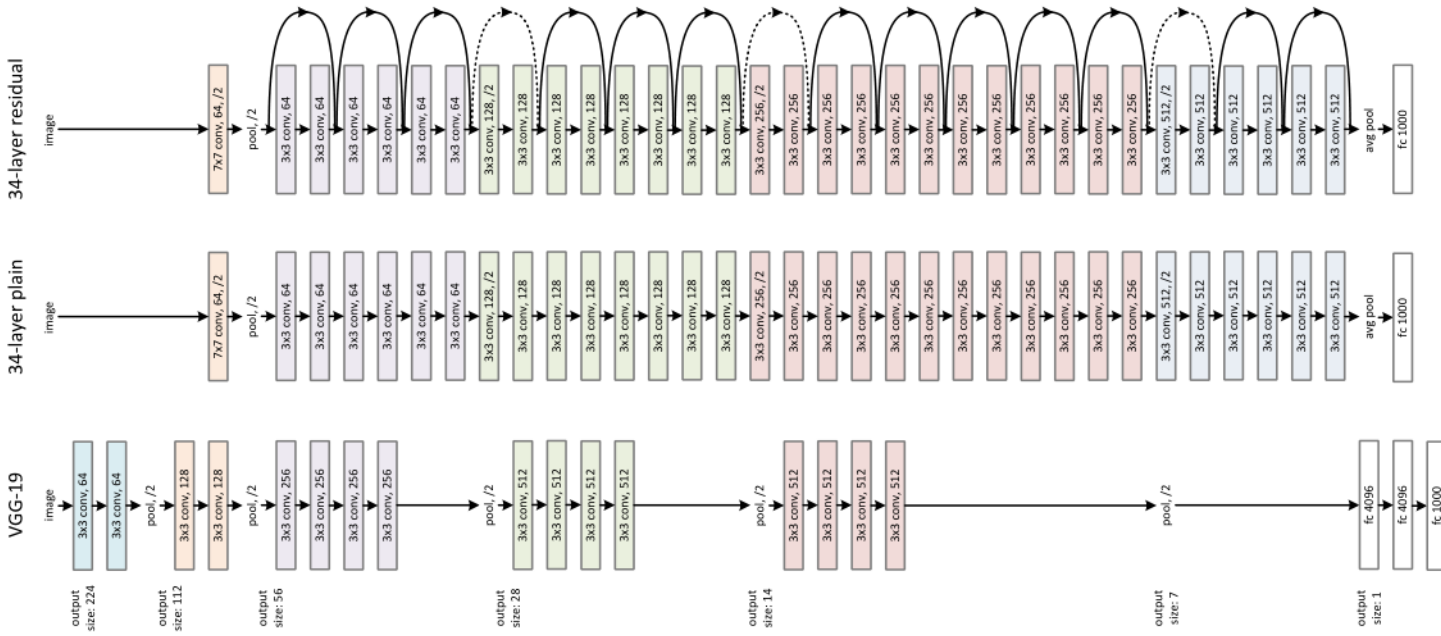
Source: He

- <u>Idea</u>: Use skip connections to help the input signal to propagate in deep networks.

- Winner of ILSVRC 2015 (top-5 error of 3.57%)

6