Deep Learning
and
Deep Reinforcement Learning

Topics

1. Deep learning with convolutional neural networks

2. Learning to play Atari video games with Deep Reinforcement Learning

3. Learning to play Go with Deep Reinforcement Learning

1. Deep learning with convolutional neural networks (in computer vision)

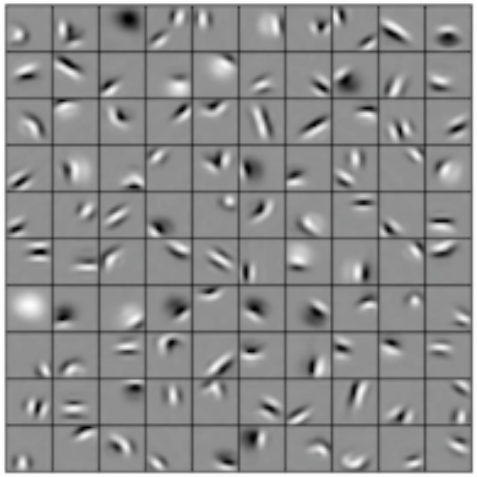
Slides adapted from Richard Turner, Cambridge University

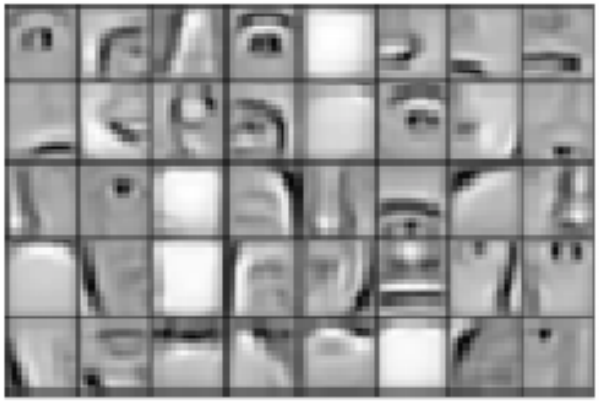
http://learning.eng.cam.ac.uk/pub/Public/Turner/Teaching/ml-lecture-3-slides.pdf

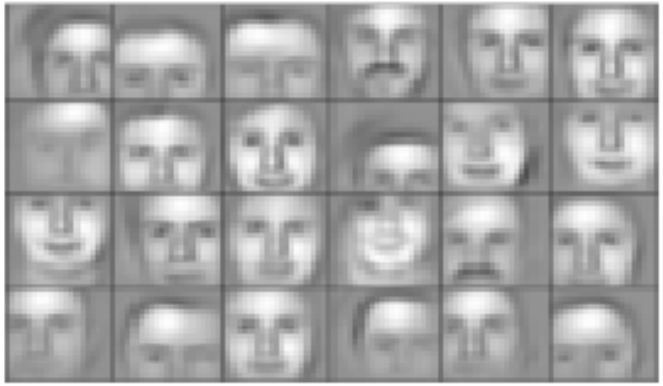
Goal

- Learn to create good, hierarchical features for visual input!

- Such features improve classification



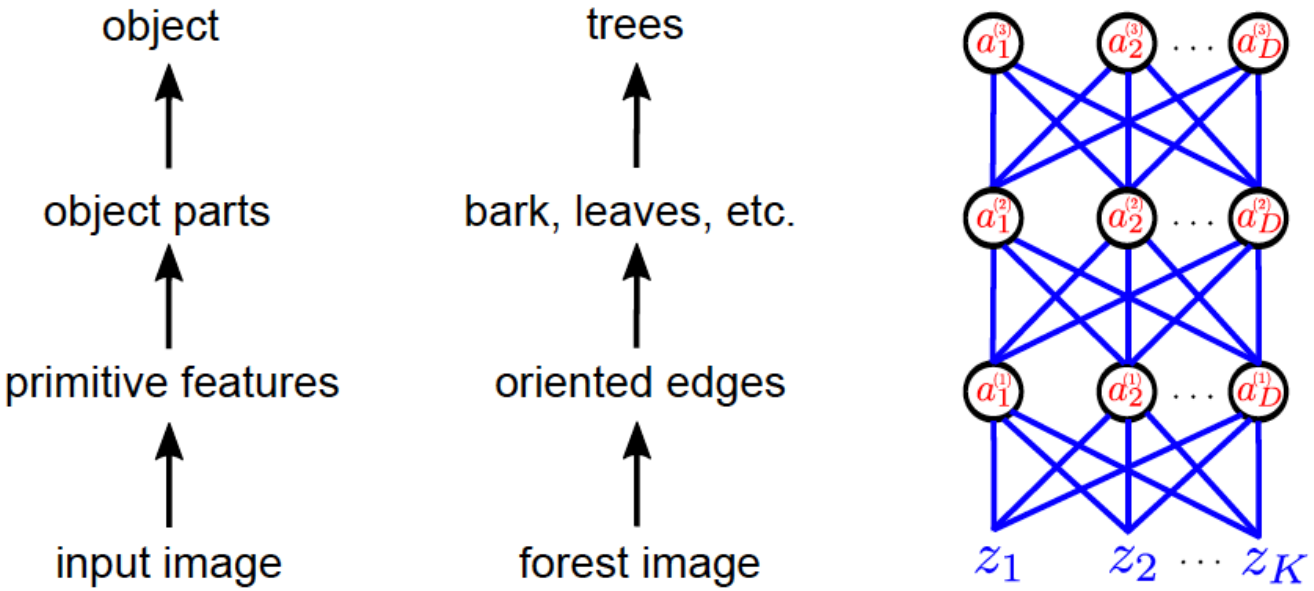




- Convolutional networks learn to generate good features

Why use hierarchical multi-layered models?

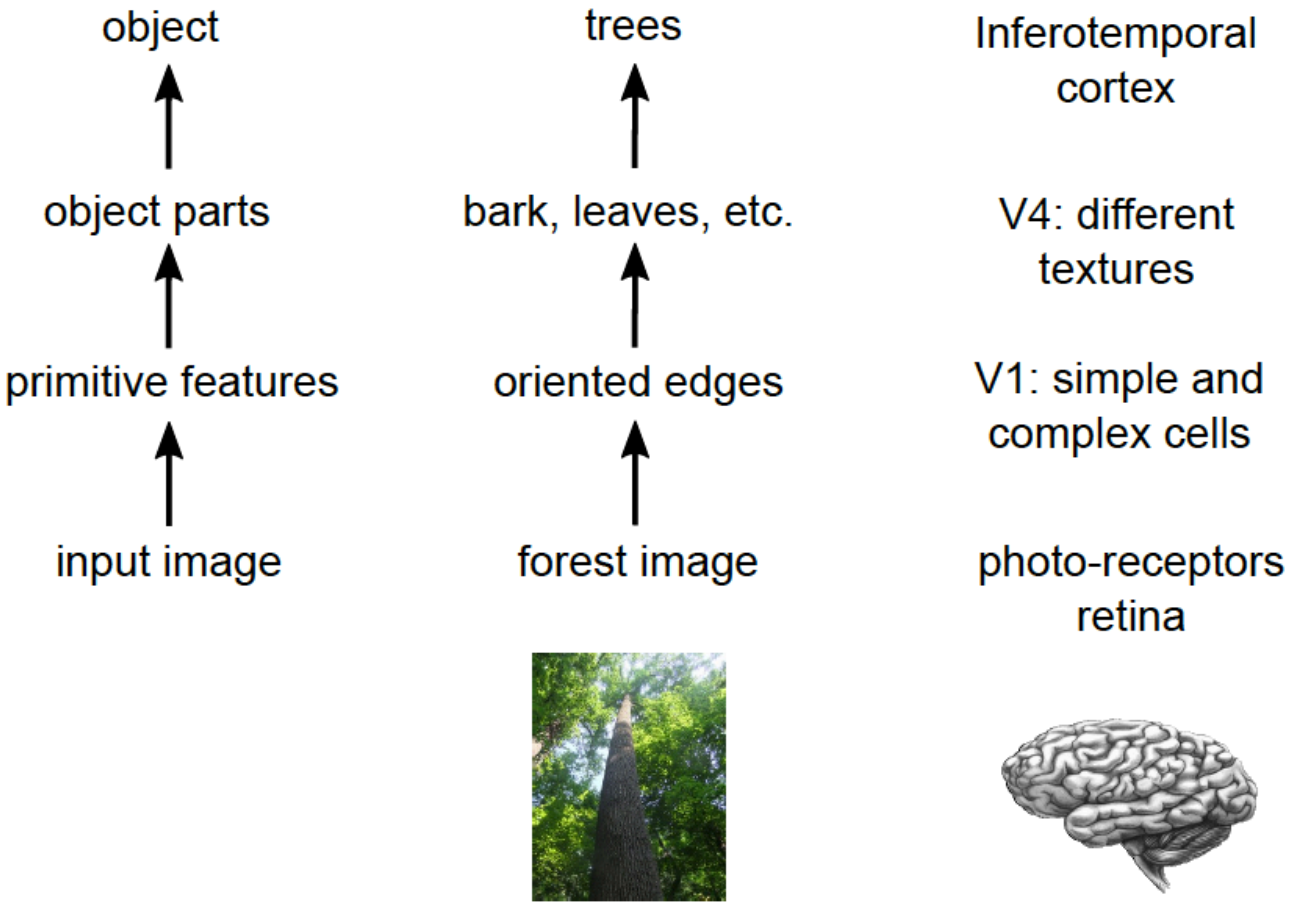
Argument 1: visual scenes are hierachically organised



From *http://learning.eng.cam.ac.uk/pub/Public/Turner/Teaching/ml-lecture-3-slides.pdf*

Why use hierarchical multi-layered models?

Argument 2: biological vision is hierachically organised

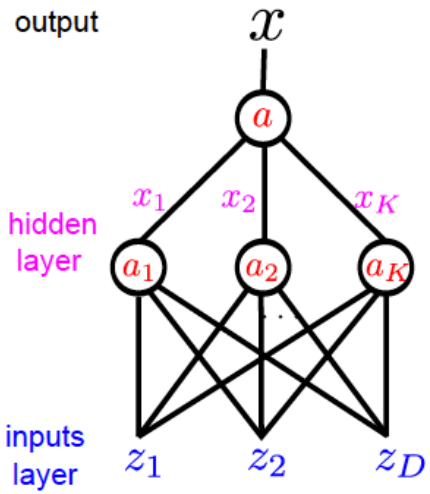


From *http://learning.eng.cam.ac.uk/pub/Public/Turner/Teaching/ml-lecture-3-slides.pdf*

Why use hierarchical multi-layered models?

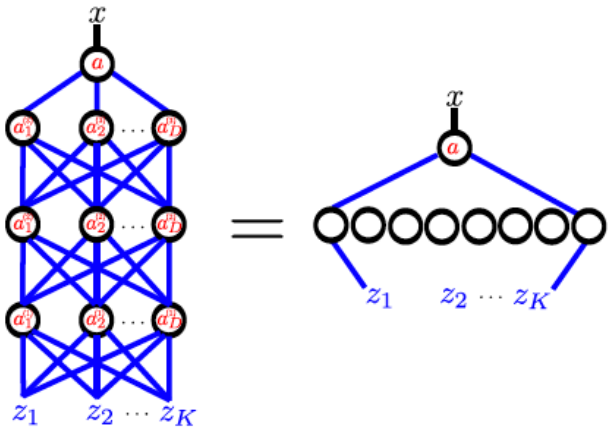
Argument 3: shallow architectures are inefficient at representing deep functions

single layer neural network implements: $x = f_\theta(\mathbf{z})$



networks we met last lecture with large enough single hidden layer can implement **any** function '**universal approximator**'

shallow networks can be computationally inefficient



however, if the function is 'deep' a very large hidden layer may be required

From *http://learning.eng.cam.ac.uk/pub/Public/Turner/Teaching/ml-lecture-3-slides.pdf*

What's wrong with standard neural networks?

How many parameters does this neural network have?

$$|\theta| = 3D^2 + D$$

For a small 32 by 32 image:

$$|\theta| = 3 \times 32^4 + 32^2 \approx 3 \times 10^6$$

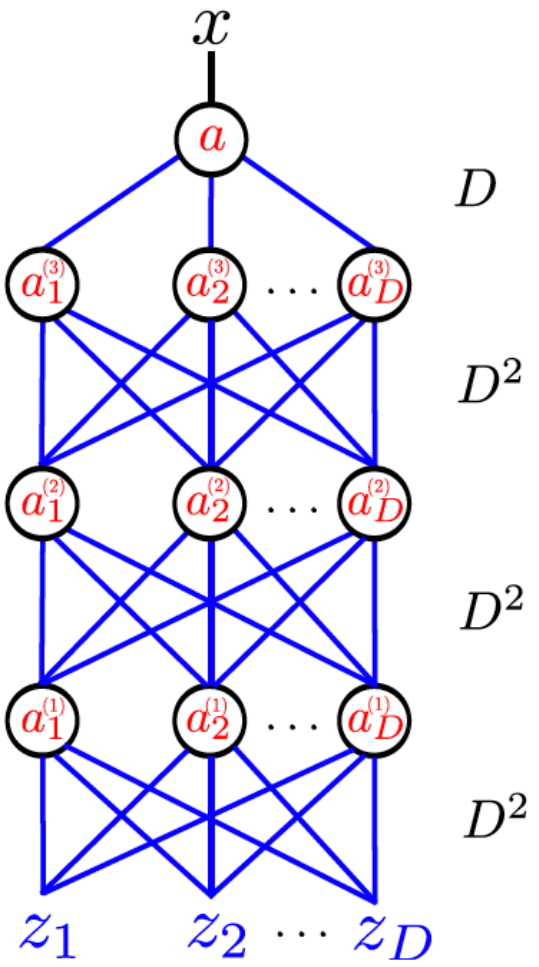
Hard to train

over-fitting and local optima

Need to initialise carefully

layer wise training
unsupervised schemes

Convolutional nets reduce the number of parameters



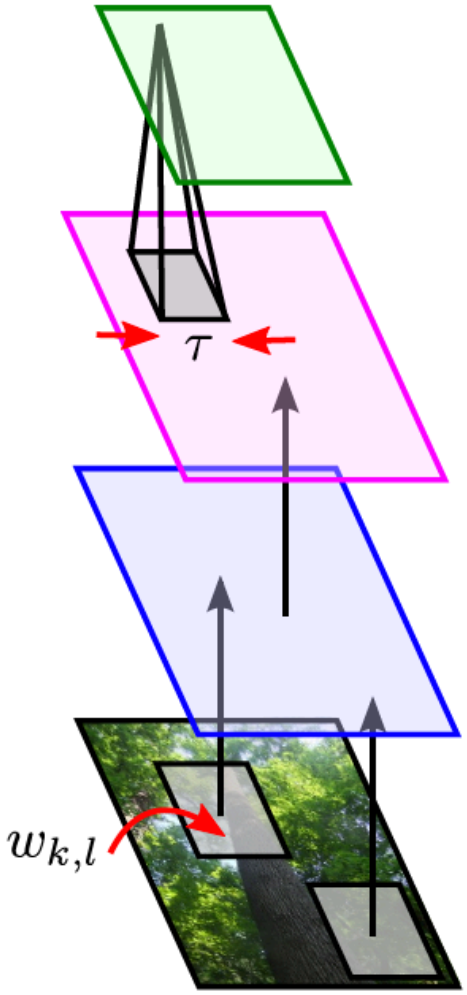
From *http://learning.eng.cam.ac.uk/pub/Public/Turner/Teaching/ml-lecture-3-slides.pdf*

The key ideas behind convolutional neural networks

- **image statistics are translation invariant** (objects and viewpoint translates)
  - build this translation invariance into the model (rather than learning it)
  - tie lots of the weights together in the network
  - reduces number of parameters
- **expect learned low-level features to be local** (e.g. edge detector)
  - build this into the model by allowing only local connectivity
  - reduces the numbers of parameters further
- **expect high-level features learned to be coarser** (c.f. biology)
  - build this into the model by subsampling more and more up the hierarchy
  - reduces the number of parameters again

From *http://learning.eng.cam.ac.uk/pub/Public/Turner/Teaching/ml-lecture-3-slides.pdf*

Building block of a convolutional neural network



$$x_{i,j} = \max_{|k|<\tau, |l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

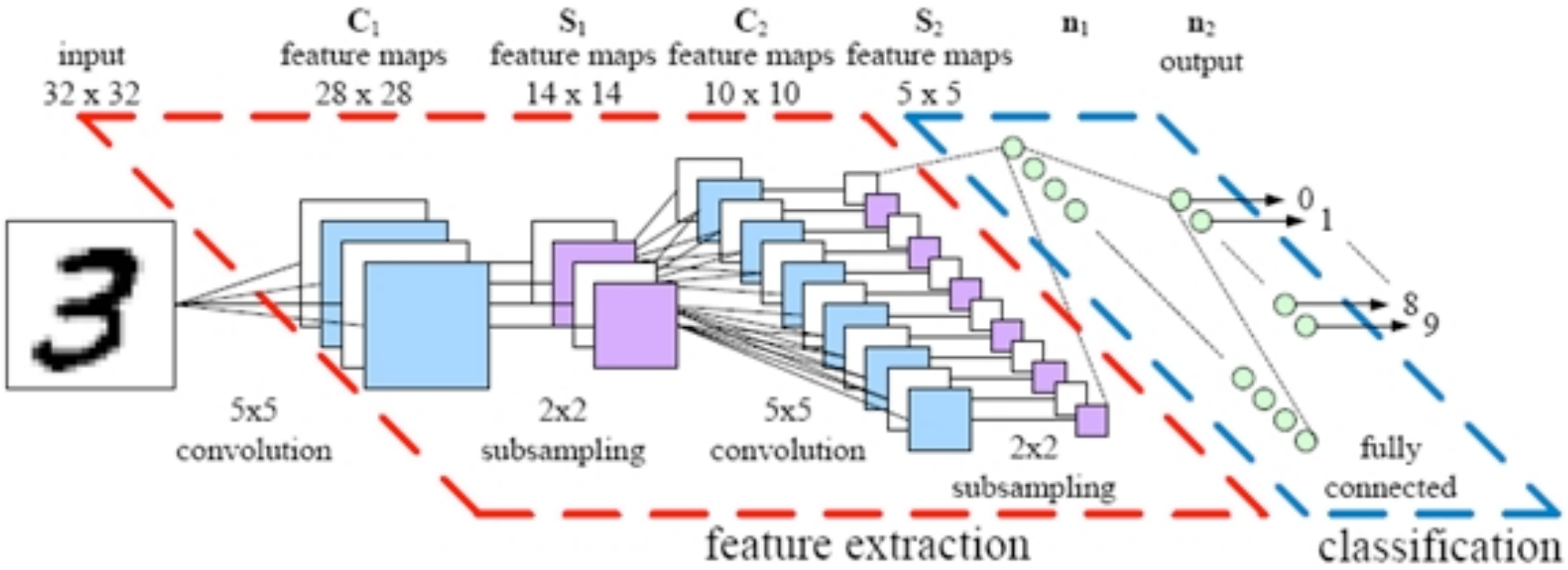
only parameters

convolutional stage

$z_{i,j}$

input image

From *http://learning.eng.cam.ac.uk/pub/Public/Turner/Teaching/ml-lecture-3-slides.pdf*



http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg

Training

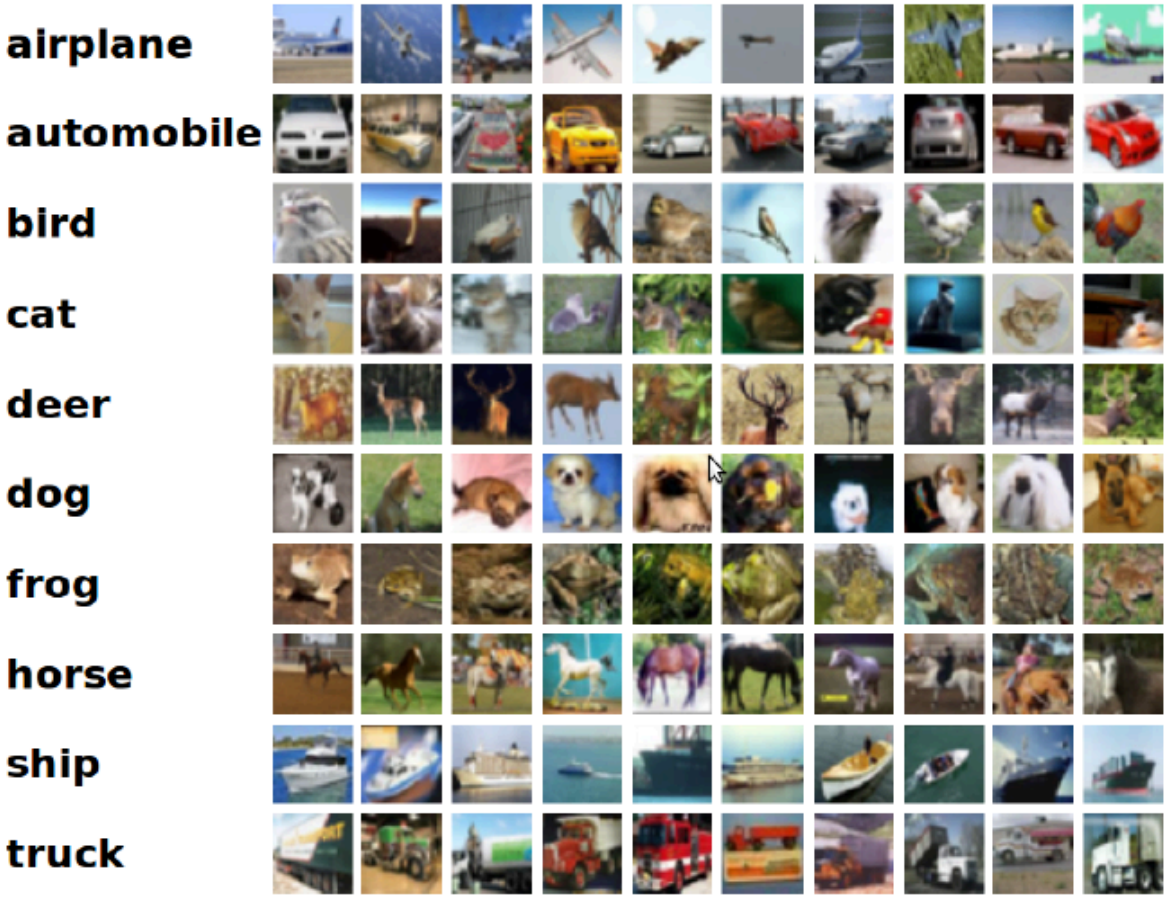
- **back-propagation for training**: stochastic gradient ascent
  - like last lecture output interpreted as a class label probability, $x = p(t = 1|\boldsymbol{z})$
  - now x is a more complex function of the inputs $\boldsymbol{z}$
  - can optimise same objective function computed over a mini-batch of datapoints

- **data-augmentation**: always improves performance substantially (include shifted, rotations, mirroring, locally distorted versions of the training data)

- **typical numbers**:
  - 5 convolutional layers, 3 layers in top neural network
  - 500,000 neurons
  - 50,000,000 parameters
  - 1 week to train (GPUs)

From *http://learning.eng.cam.ac.uk/pub/Public/Turner/Teaching/ml-lecture-3-slides.pdf*

Demo

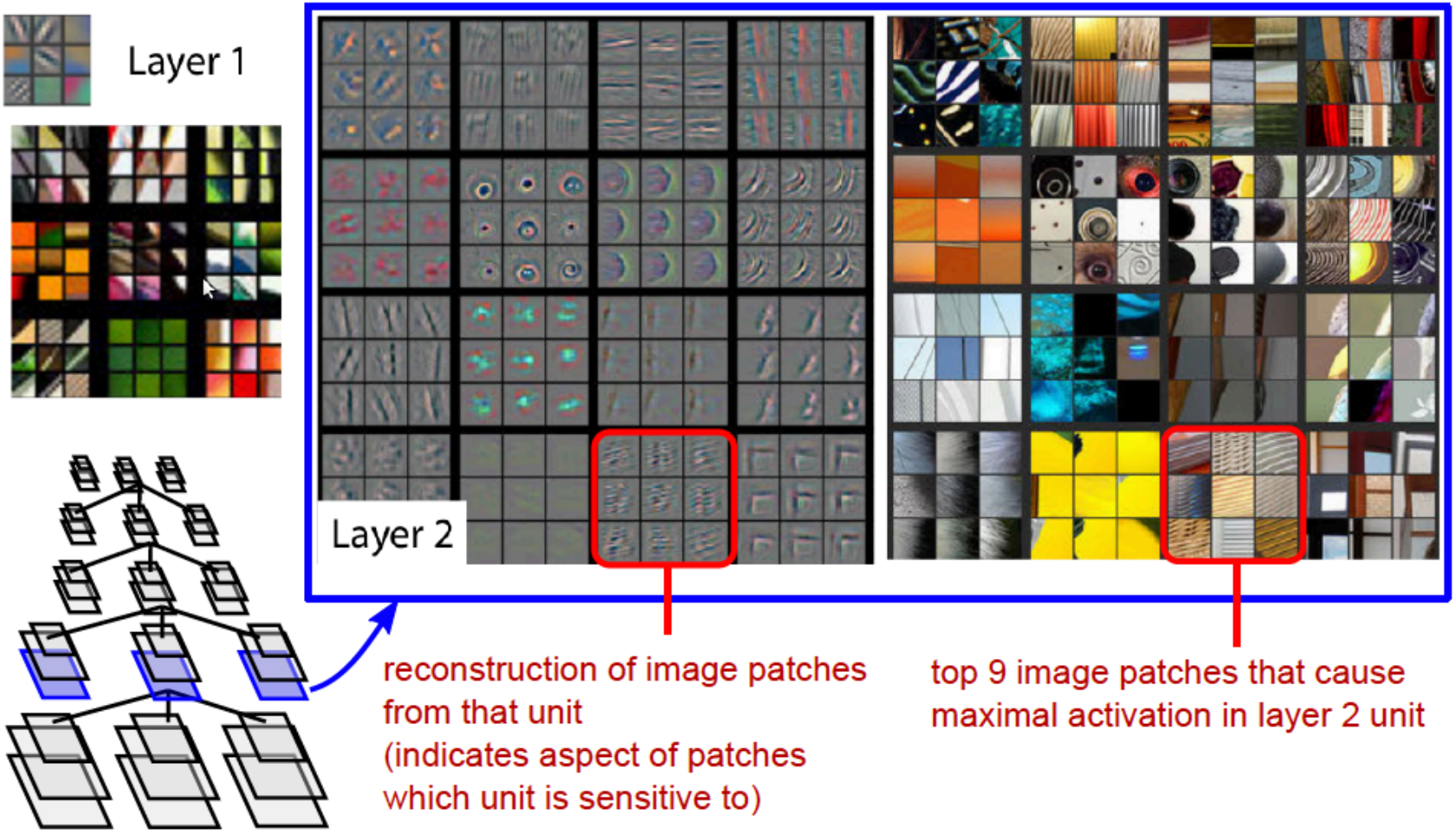


CIFAR 10 dataset: 50,000 training images, 10,000 test images

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

From *http://learning.eng.cam.ac.uk/pub/Public/Turner/Teaching/ml-lecture-3-slides.pdf*

Looking into a convolutional neural network's brain

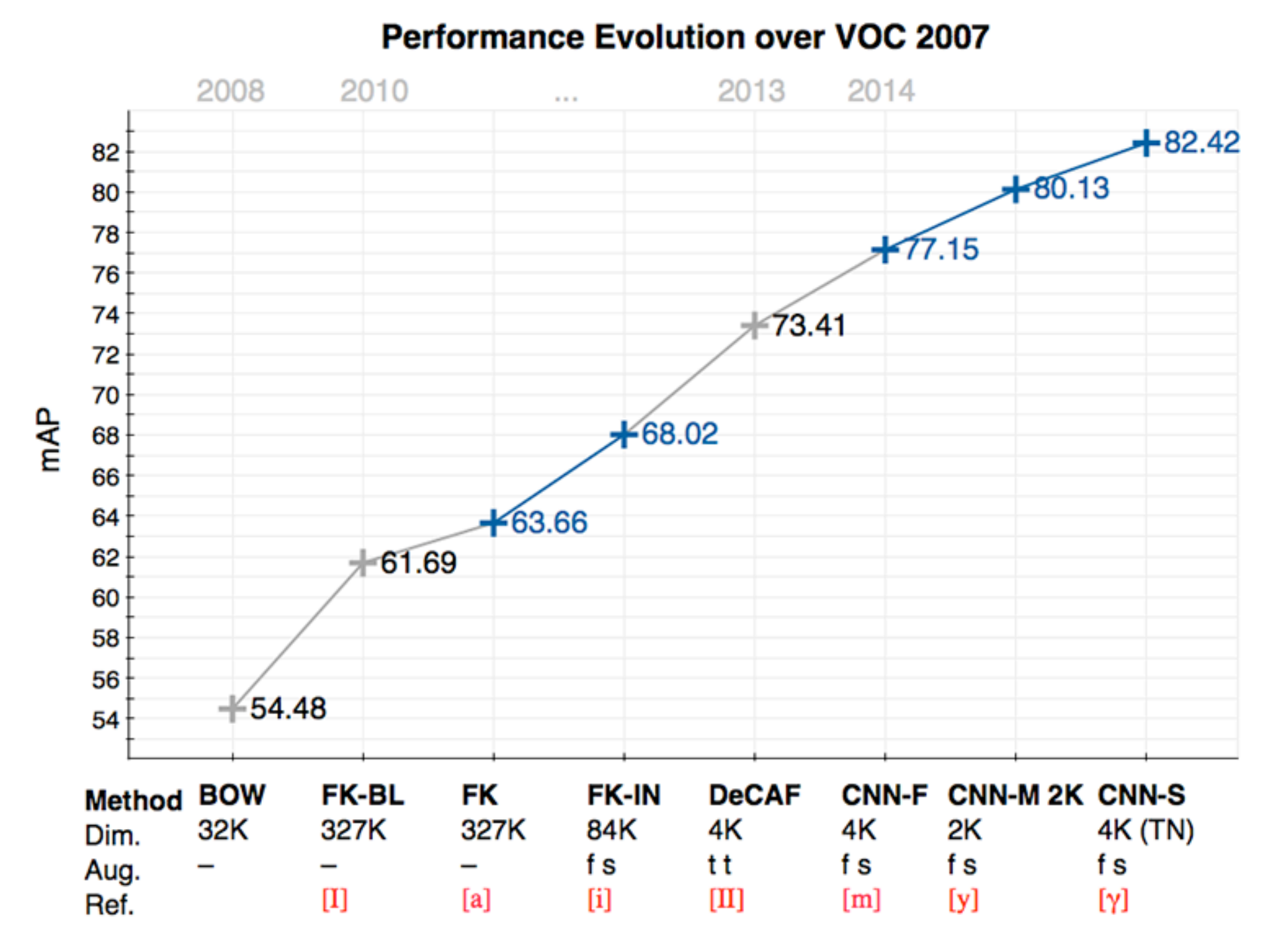


From *http://learning.eng.cam.ac.uk/pub/Public/Turner/Teaching/ml-lecture-3-slides.pdf*

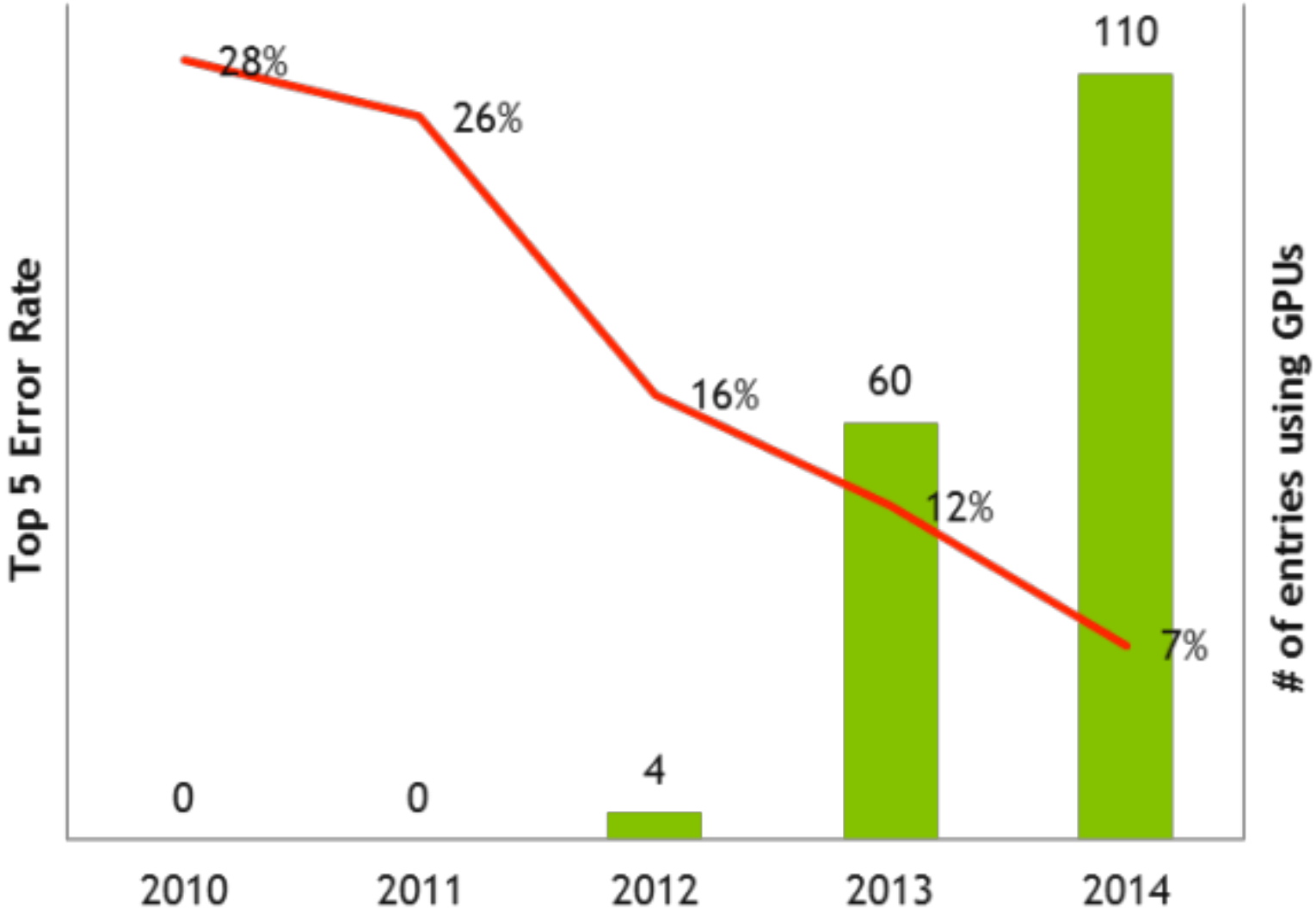
Summary

- higher level layers encode more **abstract features**
- higher level layers show more **invariance to instantiation parameters**
  - translation
  - rotation
  - lighting changes
- a method for **learning feature detectors**
  - first layer learns edge detectors
  - subsequent layers more complex
  - integrates training of the classifier with training of the featural representation

From *http://learning.eng.cam.ac.uk/pub/Public/Turner/Teaching/ml-lecture-3-slides.pdf*


Performance Evolution over VOC 2007
2008
2010
...
2013
2014
mAP
82
80
78
76
74
72
70
68
66
64
62
60
58
56
54
54.48
61.69
63.66
68.02
73.41
77.15
80.13
82.42
Method BOW FK-BL FK FK-IN DeCAF CNN-F CNN-M 2K CNN-S
Dim. 32K 327K 327K 84K 4K 4K 2K 4K (TN)
Aug. – – – f s t t f s f s f s
Ref. [I] [a] [i] [II] [m] [y] [γ]

IMAGENET


Top 5 Error Rate
# of entries using GPUs
28%
26%
16%
12%
7%
0
0
4
60
110
2010
2011
2012
2013
2014

“Deep Neural Networks are Easily Fooled”

Nguyen et al., 2015

https://www.youtube.com/watch?v=M2IebCN9Ht4

Learning to play Atari video games with Deep Reinforcement Learning

Mnih et al., DeepMind Technologies, 2013

Video: https://www.youtube.com/watch?v=5WXVJ1A0k6Q

- Deep learning:
  - Requires large amount of hand-labeled data
  - Assumes data samples are independent, with stationary distribution
- Reinforcement learning:
  - Must learn from sparse, noisy, delayed reward function
  - "Samples" are not independent
  - Data distribution changes as system learns "online".

2. Learning to play Atari video games with Deep Reinforcement Learning

Mnih et al., DeepMind Technologies, 2013

- Uses CNN:
  - Input is raw pixels of video frames (~ the "state")

- Output is estimated Q(s,a) for each possible action

- To alleviate problem of correlated data and non-stationary distributions, use **"experience reply" mechanism**, which randomly samples previous transitions, and "thereby smooths the training distribution over many past behaviors."

- Their system learns to play Atari 2600 games.
- 210x160 RGB video at 60 Hz.
- Designed to be difficult for human players
- “Our goal is to create a single neural network agent that is able to successfully learn to play as many of the games as possible.”
- No game-specific info provided. No hand-designed visual features.
- Learns exclusively from video input, the reward, and terminal signals. Network architecture and hyperparameters kept constant across all games.

- CNN: Input is set of 4 most recent video frames, output vector is Q-value of each possible action, given the input state
- At each time step t, agent selects action a_t from set of legal actions $A = \{1, \ldots, K\}$.
- Action a_t is passed to emulator.
- Game state and current score are updated.
- Agent observes image x_t from emulator (vector of raw pixels representing current screen)
- Agent receives reward r_t representing change in game score. (Usually 0.)

- $Q(s,a \mid \pi)$ = expected return achievable, following policy π after seeing some sequence s and then taking some action a.

- Do TD-learning as in backgammon:
  - Train CNN with loss function:

$$L_t(\theta_t) = \left(\gamma \max_{a'} Q(s', a' \mid \theta_t) - Q(s, a \mid \theta_t)\right)$$

where θ_t is the current set of weights in the network, s is the current state, a is the selected action, s' is the state that action a leads to, and γ is the future discounting factor.

- Minimize loss function via stochastic gradient descent on θ_t (weights)

- TD gammon was successful for backgammon. People tried to use it for chess, go, checkers, but not as successful.

- This work: Combines deep neural networks with TD learning.

- "Experience replay" technique: At each time step, store $e_t = (s_t , a_t , r_t , s_{t+1})$

- Training data set is $(e_1 , e_2 , ..., e_N)$

- Apply Q-learning updates using samples of experience e, drawn at random from the training data.

- Doing this avoids overfitting to particular game state sequence

output of CNN

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]. \quad (3)$$

Advantages over Q-learning

- Each step of experience potentially used in many weight updates.
- Learning directly from consecutive samples is inefficient, due to strong correlations between samples. Randomizing the samples breaks these correlations.
- Note "off-policy" learning (current parameters different from those used to generate the sample)
- Uniform sampling of experience gives equal importance to all transitions – this could be a bad thing…

Experiments

- Used same CNN architecture, learning algorithm, and parameters across seven Atari games.

Video

https://www.youtube.com/watch?v=Ih8EfvOzBOY

3. Mastering the game of Go with deep neural networks and tree search

Silver et al., 2016

https://www.youtube.com/watch?v=SUbqykXVx0A

3. Mastering the game of Go with deep neural networks and tree search

Silver et al., 2016

- Go, like chess, is a finite, deterministic game of "perfect information":
  - If you could look ahead far enough, you could play optimal moves.
  - In principle there is a value function V*(s) that gives true value of a board state s
- In practice, search tree is too big!! Much bigger than chess.

Alpha Go

- Puts together several existing methods:
  - Policy network (CNN) that learns from expert human moves in a supervised manner
  - Policy network (CNN) that starts with weights from supervised network, but learns further via reinforcement learning under self-play
  - Value network (CNN) for estimating $V^*(sa)$ from these features
  - Monte Carlo tree search for sampling game tree

Training pipeline

- **Supervised policy network** $p_\sigma(a|s)$:
  - Learns to predict expert moves from board states, using supervised learning (from database of expert games).
  - p_σ is learned via a 13-layer CNN.
  - Inputs board state; Outputs probability distribution over actions.
  - Weights updated at each time step by stochastic gradient descent in direction that minimizes difference between output distribution and target distribution.

- We also trained a faster but less accurate rollout policy $p_\pi(a|s)$, using a linear softmax of small pattern features

Extended Data Table 4 | Input features for rollout and tree policy

Feature	# of patterns	Description
Response	1	Whether move matches one or more response pattern features
Save atari	1	Move saves stone(s) from capture
Neighbour	8	Move is 8-connected to previous move
Nakade	8192	Move matches a *nakade* pattern at captured stone
Response pattern	32207	Move matches 12-point diamond pattern near previous move
Non-response pattern	69338	Move matches 3×3 pattern around move
Self-atari	1	Move allows stones to be captured
Last move distance	34	Manhattan distance to previous two moves
Non-response pattern	32207	Move matches 12-point diamond pattern centred around move

Features used by the rollout policy (first set) and tree policy (first and second set). Patterns are based on stone colour (black/white/empty) and liberties (1, 2, ≥3) at each intersection of the pattern.

Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Feature planes used by the policy network (all but last feature) and value network (all features).

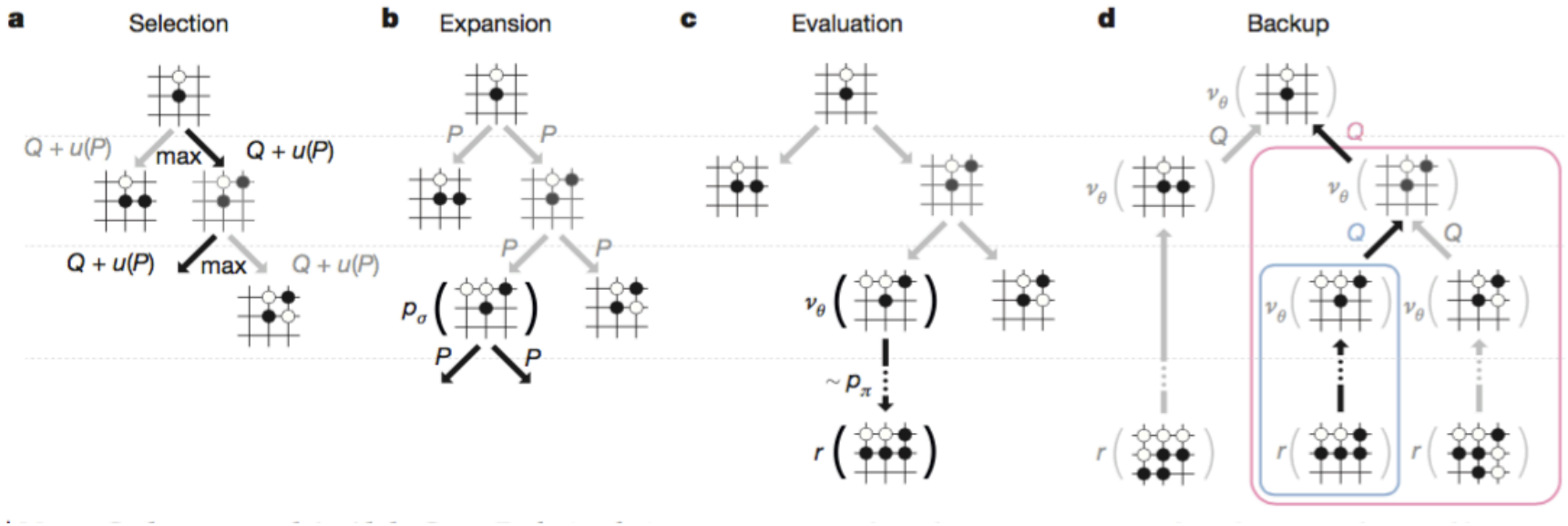
- **RL policy network** $p_\rho(a|s)$:
  - Identical in structure to supervised policy network.
  - Weights initialized to weights of p_σ .
  - Inputs current state; Outputs probability distribution over actions.
  - Learns by playing games between current p_σ and randomly selected previous iteration of p_σ .
  - This random selection prevents overfitting to current policy
  - Reward function is zero for all non-terminal time steps. Final reward is game outcome (+1 or -1) from point of view of current player.
  - Weights updated at each time step by stochastic gradient ascent in direction that maximizes expected outcome.

- In head-to-head games, RL policy network won more than 80% of games against the supervised learning policy network.

- Similarly, RL policy network won 85% of games against best current Go program (that uses Monte Carlo tree search).

- **RL value network** $v^p(s)$:
  - Estimates value of state s when using policy p for both players
  - Use RL policy network p_ρ for policy p
  - v^p (CNN): inputs state s, outputs single value (predicting outcome from s)
  - Weights of v^p are trained using regression on state-outcome pairs, using stochastic gradient descent to minimize mean-squared error between predicted value and corresponding outcome.
  - Training data comes from 30 million distinct positions sampled from different self-play games (using RL policy network).

Monte Carlo Tree Search



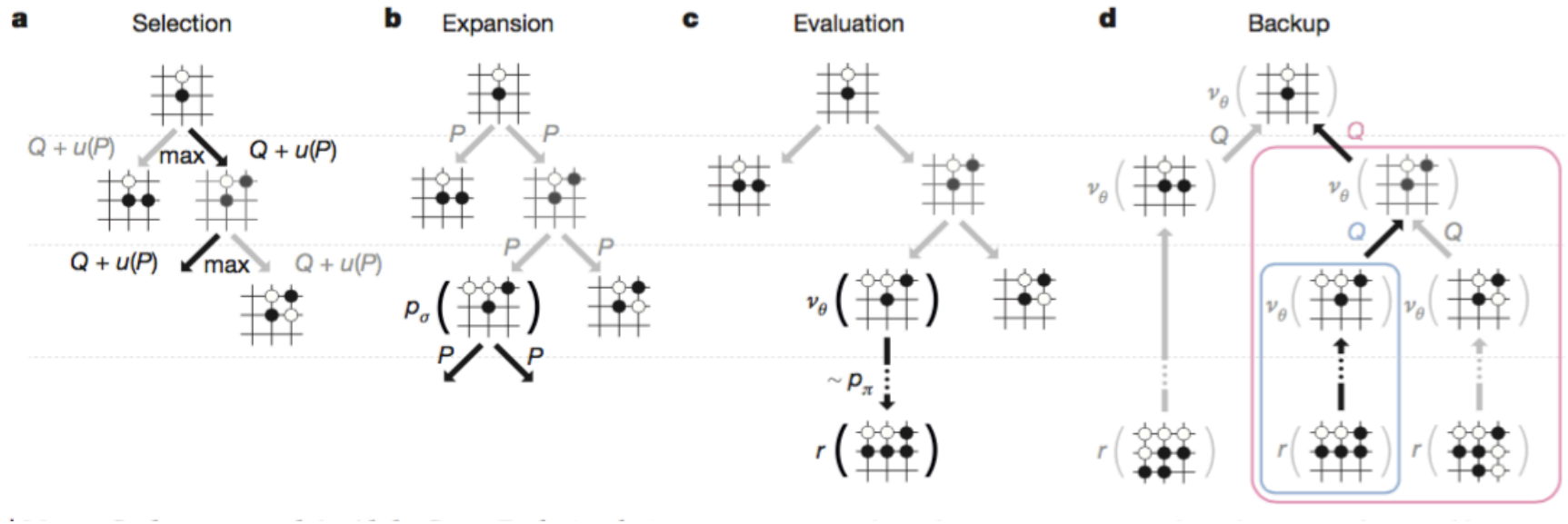
Each simulation (look ahead episode):

(a) Select edge with maximum action value Q, plus a bonus $u(P)$ that depends on stored prior probability P for that edge

- P is a function of p_σ / # of prior visits

The denominator encourages exploration.

Monte Carlo Tree Search

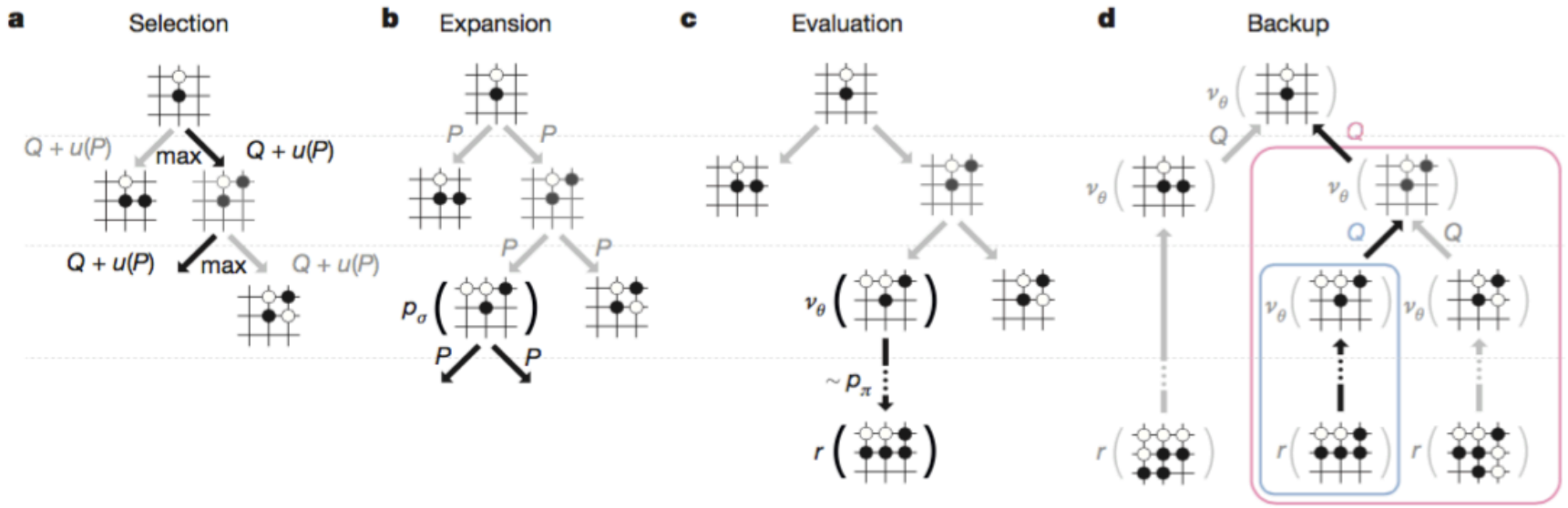


(b) Expand leaf nodes:

- Each leaf node is processed by policy network p_σ . The output probabilities are stored as prior probabilities P for each action.

- # of visits to this node is incremented.

Monte Carlo Tree Search

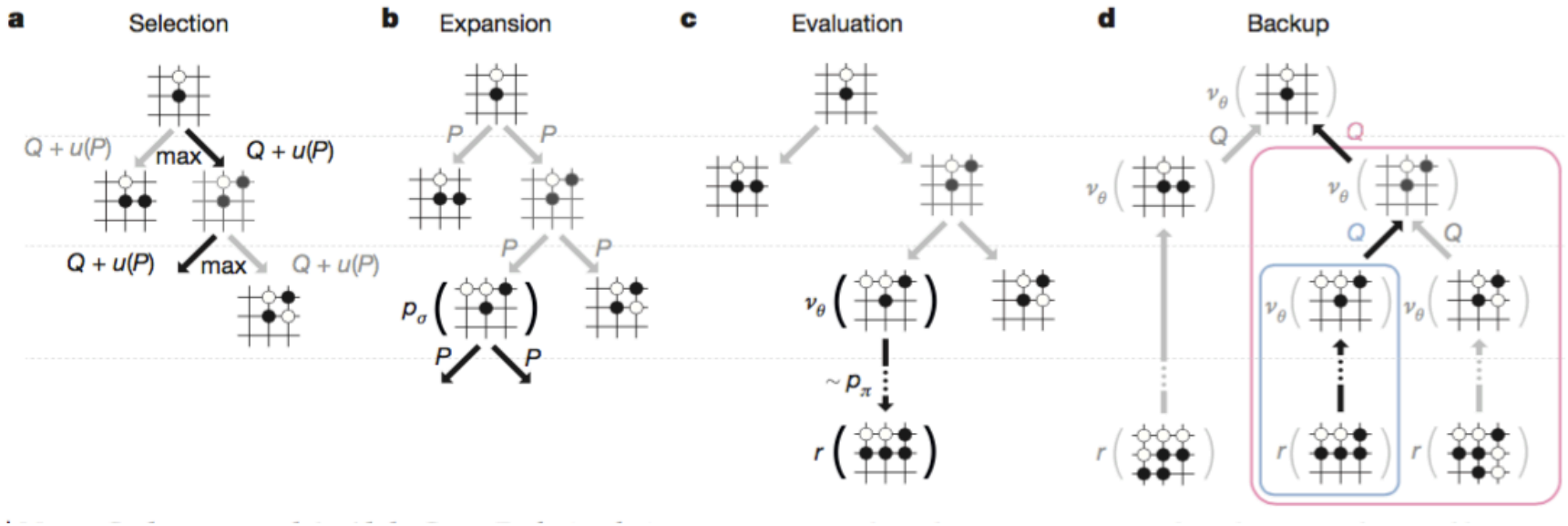


(c) At end of simulation, the final leaf node is evaluated using:
(1) current value network v_θ
(2) running rollout to end of game with fast rollout policy p_π .

- Value is a weighted combination of these two values:

$$V(s_L) = (1-\lambda)\, v_\theta(s_L) + \lambda z_L, \quad where\ z_L = \begin{cases} 1 \text{ if outcome of fast rollout is win by this player} \\ -1 \text{ otherwise} \end{cases}$$

Monte Carlo Tree Search



(d) Each edge accumulates the visit count and mean evaluation of all simulations passing through that edge.

Once search is complete, the algorithm chooses the most visited move from the root position.

a
Elo Rating
3,500
3,000
2,500
2,000
1,500
1,000
500
0
Professional dan (p)
Amateur dan (d)
Beginner kyu (k)
9p
7p
5p
3p
1p
9d
7d
5d
3d
1d
1k
3k
5k
7k
AlphaGo distributed
AlphaGo
AlphaGo
Fan Hui
Crazy Stone
Zen
Pachi
Fuego
GnuGo