An Introduction to Deep Learning II

Yuan YAO

HKUST

Visualizing NN and Transfer Learning

Visualizing Deep Neural Networks

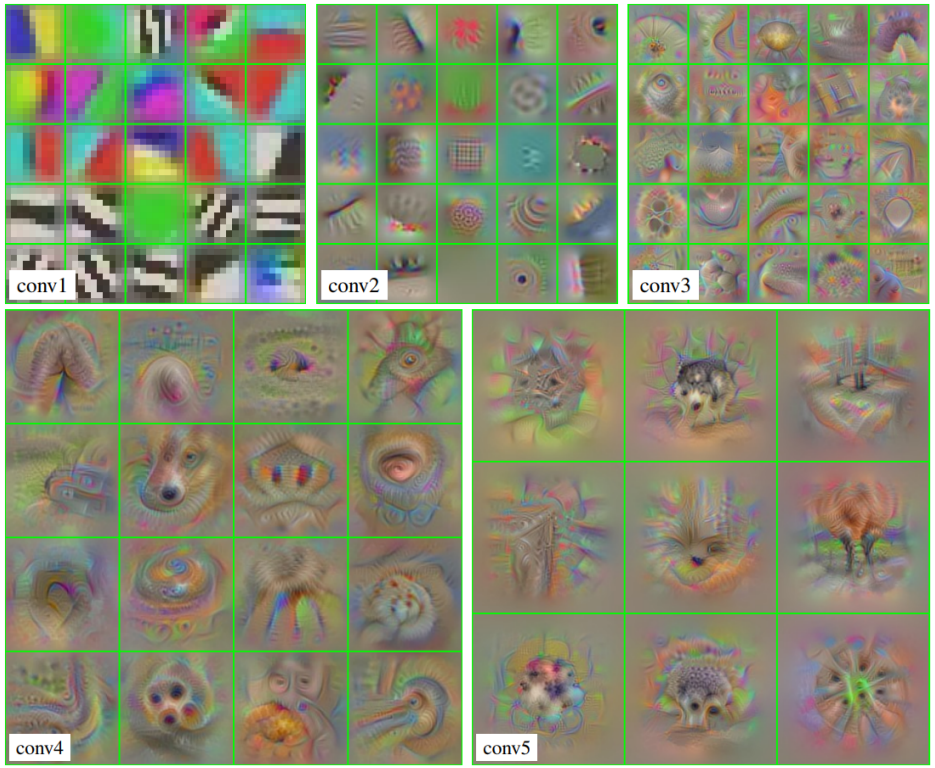
- Filters in first layer of CNN are easy to visualize, while deeper ones are harder
- *Activation maximization* seeks input image maximizing output of the i-th neuron in the network
- Objective

$$x^* = \arg\min_x \mathcal{R}(x) - \langle \Phi(x), e_i \rangle$$

- e_i is indicator vector
- $\mathcal{R}(x)$ is simple natural image prior

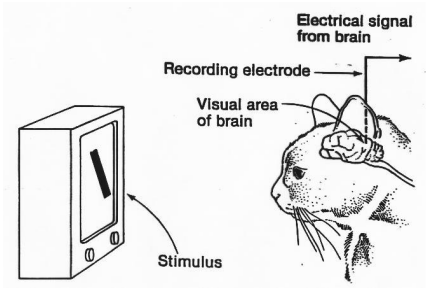
Visualizing VGG

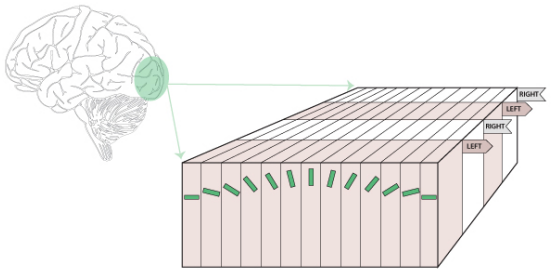
- Gabor-like images in first layer
- More sophisticated structures in the rest

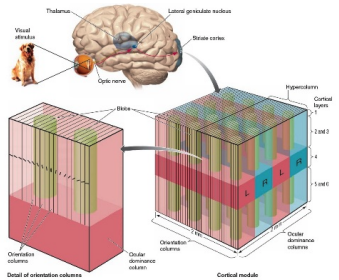


[Mahendran and Vedaldi, 2016]

Visual Neuroscience: Hubel/Wiesel, …

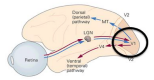


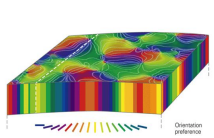




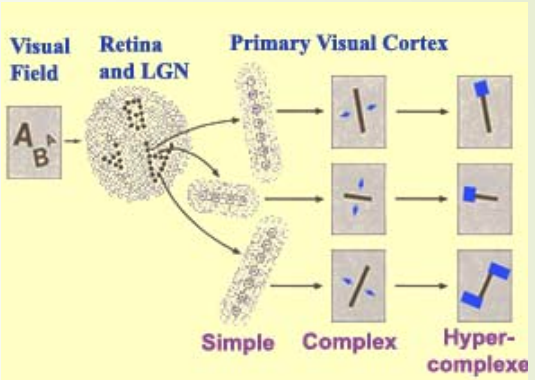
V1 orientation maps

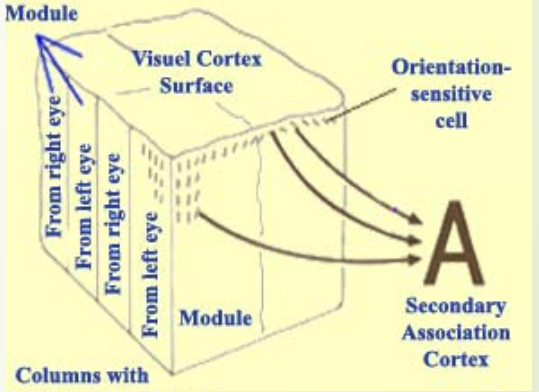
- Continuous maps of orientation preference: "pinwheels"
- Consistent preference though the depth of cortex: columnar architecture





Kandell et. al., *Princ. of Neural Sci.*





Olshausen and Field 1996

Experimental Neuroscience uncovered the

- ... neural architecture of Retina/LGN/V1/V2/V3/ etc
- ... existence of neurons with weights and activation functions (simple cells)
- ... pooling neurons (complex cells)

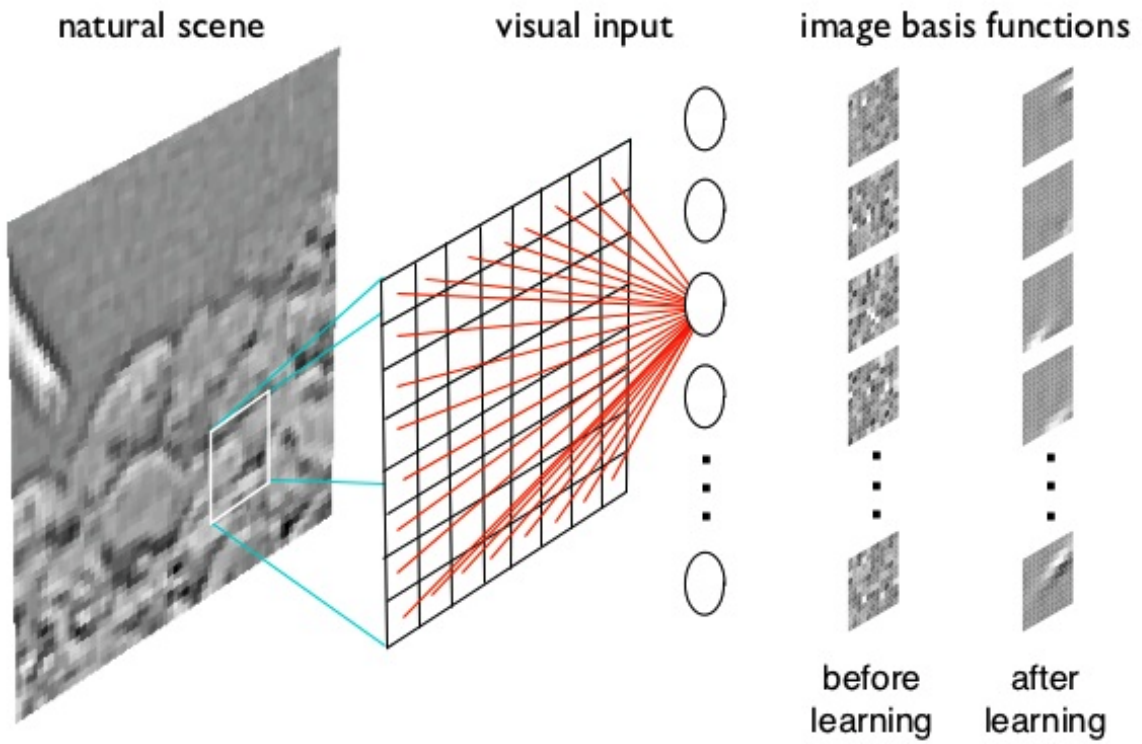
All these features are somehow present in today's sucessful Deep Learning systems

Neuroscience	Deep Network
Simple cells	First layer
Complex celle	Pooling Layer
Grandmother cells	Last layer

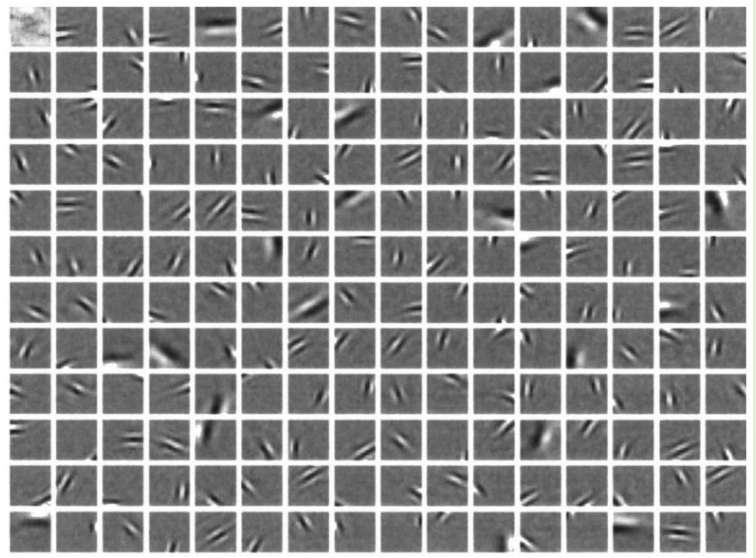
Theorists Olshausen and Field (Nature, 1996) demonstrated that receptive fields learned from image patches

First layers learned …

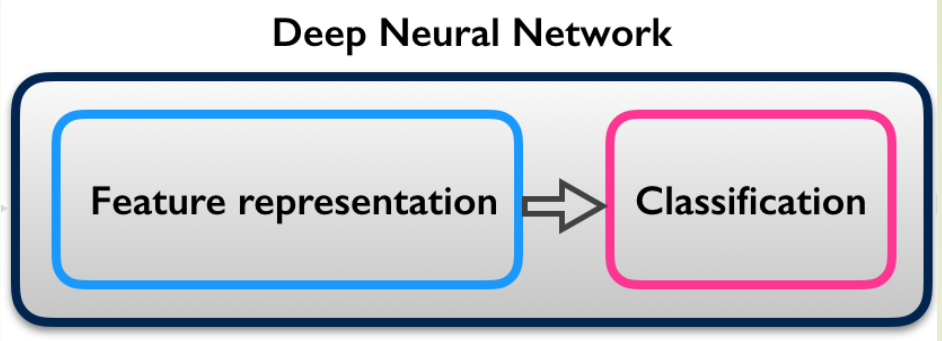
Efficient coding of natural images: Olshausen and Field, 1996



Network weights are adapted to maximize coding efficiency:
minimizes redundancy and maximizes the independence of the outputs



Transfer Learning?

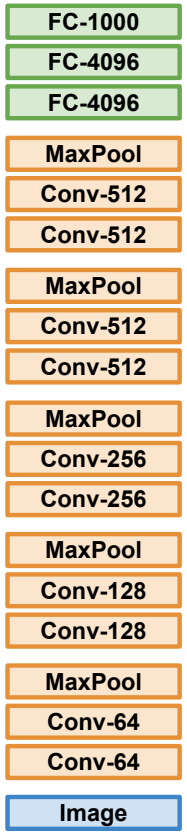


- Filters learned in first layers of a network are transferable from one task to another
- When solving another problem, no need to retrain the lower layers, just fine tune upper ones
- Is this simply due to the large amount of images in ImageNet?
- Does solving many classification problems simultaneously result in features that are more easily transferable?
- Does this imply filters can be learned in unsupervised manner?
- Can we characterize filters mathematically?

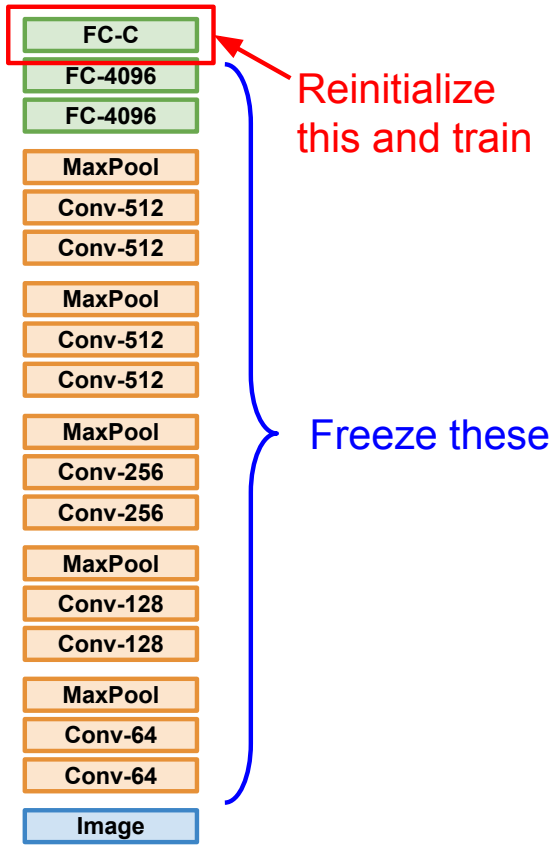
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

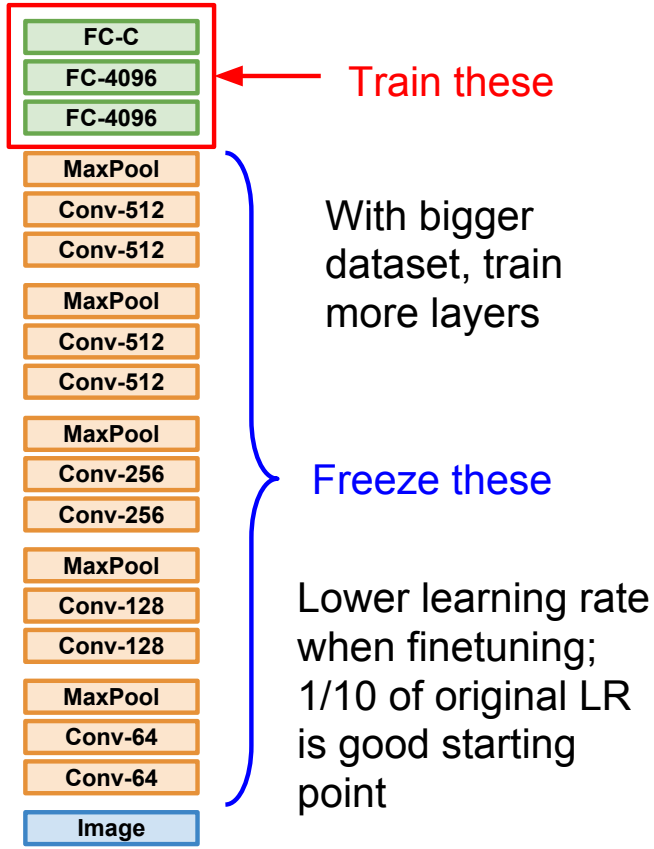
1. Train on Imagenet

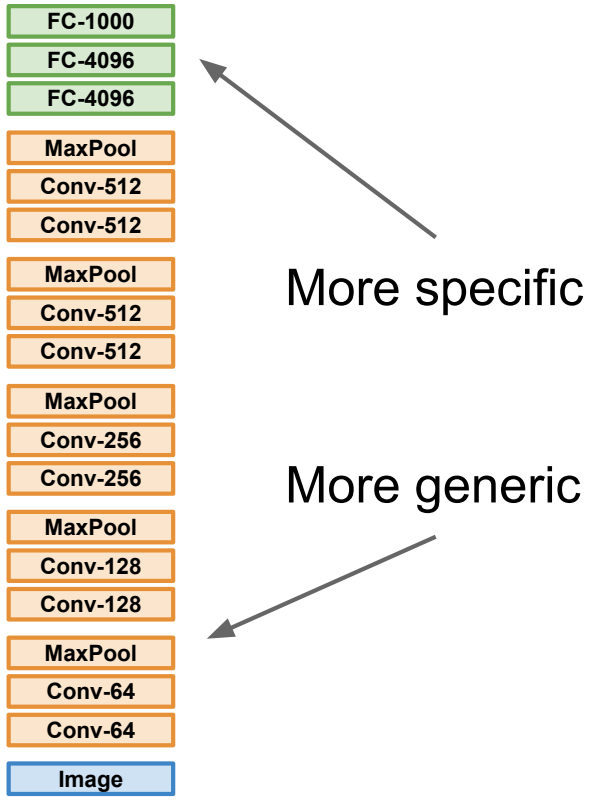


2. Small Dataset (C classes)



3. Bigger dataset





	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Takeaway for your projects and beyond:

Have some dataset of interest but it has < ~1M images?

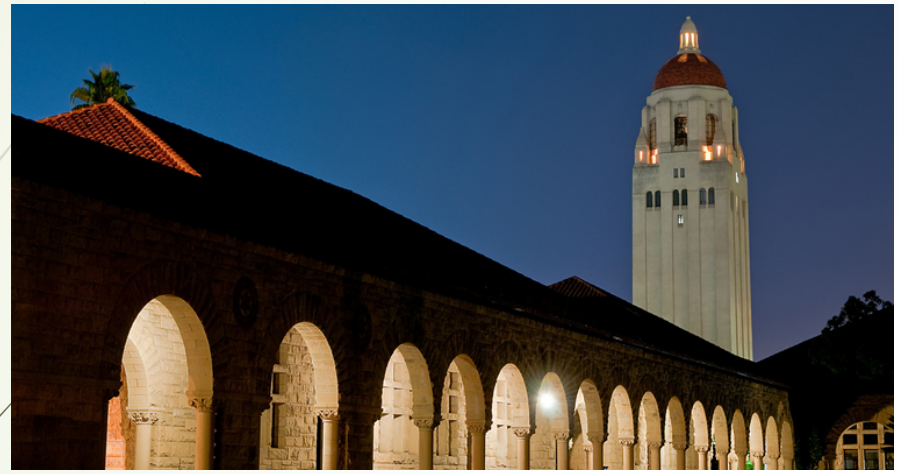
1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

Caffe: https://github.com/BVLC/caffe/wiki/Model-Zoo
TensorFlow: https://github.com/tensorflow/models
PyTorch: https://github.com/pytorch/vision

Style-Content Features

Examples

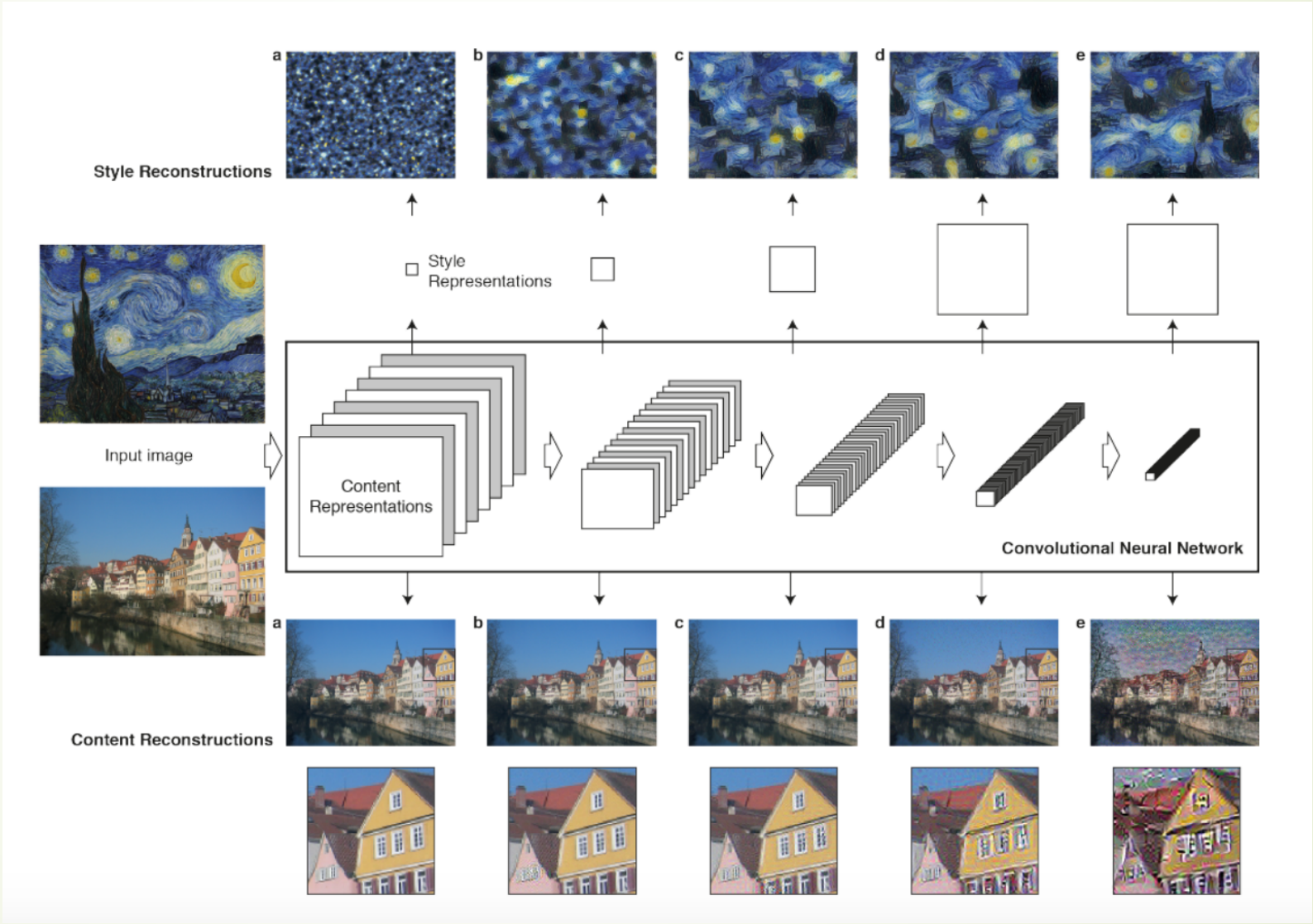


Neural Style

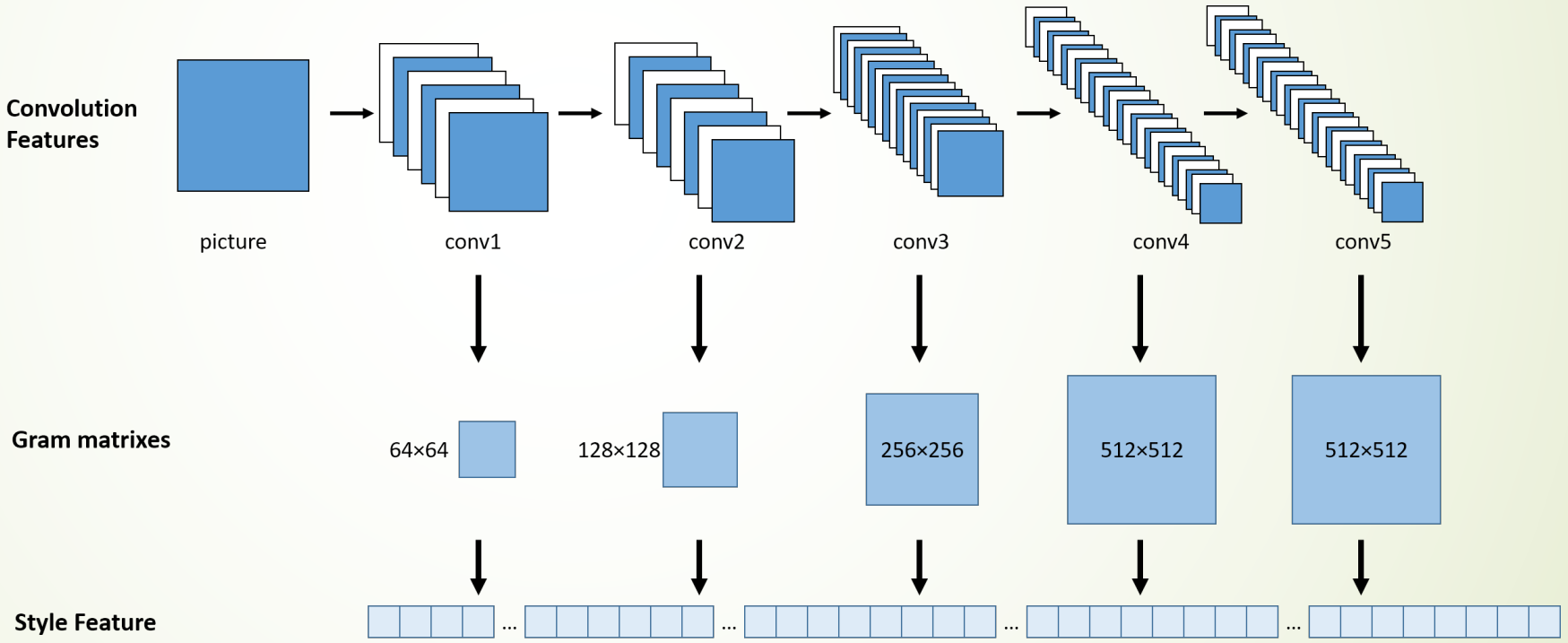
- J C Johnson's Website: https://github.com/jcjohnson/neural-style

- A torch implementation of the paper
  - *A Neural Algorithm of Artistic Style*,
  - by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge.
  - http://arxiv.org/abs/1508.06576

Style-Content Feature Extraction



Style Features as Second Order Statistics



Loss for Content and Style (2nd order statistics of outputs)

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l} \right)^2 .$$

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^{l} - A_{ij}^{l} \right)^2$$

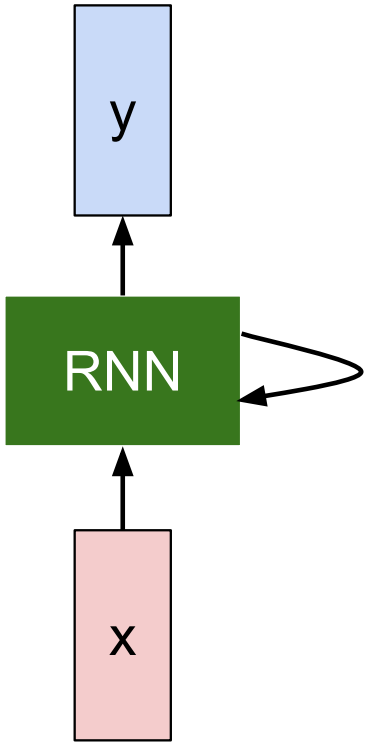
$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}.$$

Recurrent Neural Networks

Recurrent Neural Networks: sequences

Nonlinear State Space equations in feedback dynamical systems

The state consists of a single *"hidden"* vector **h**:



$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Or, $y_t = \text{softmax}(W_{hy}h_t)$

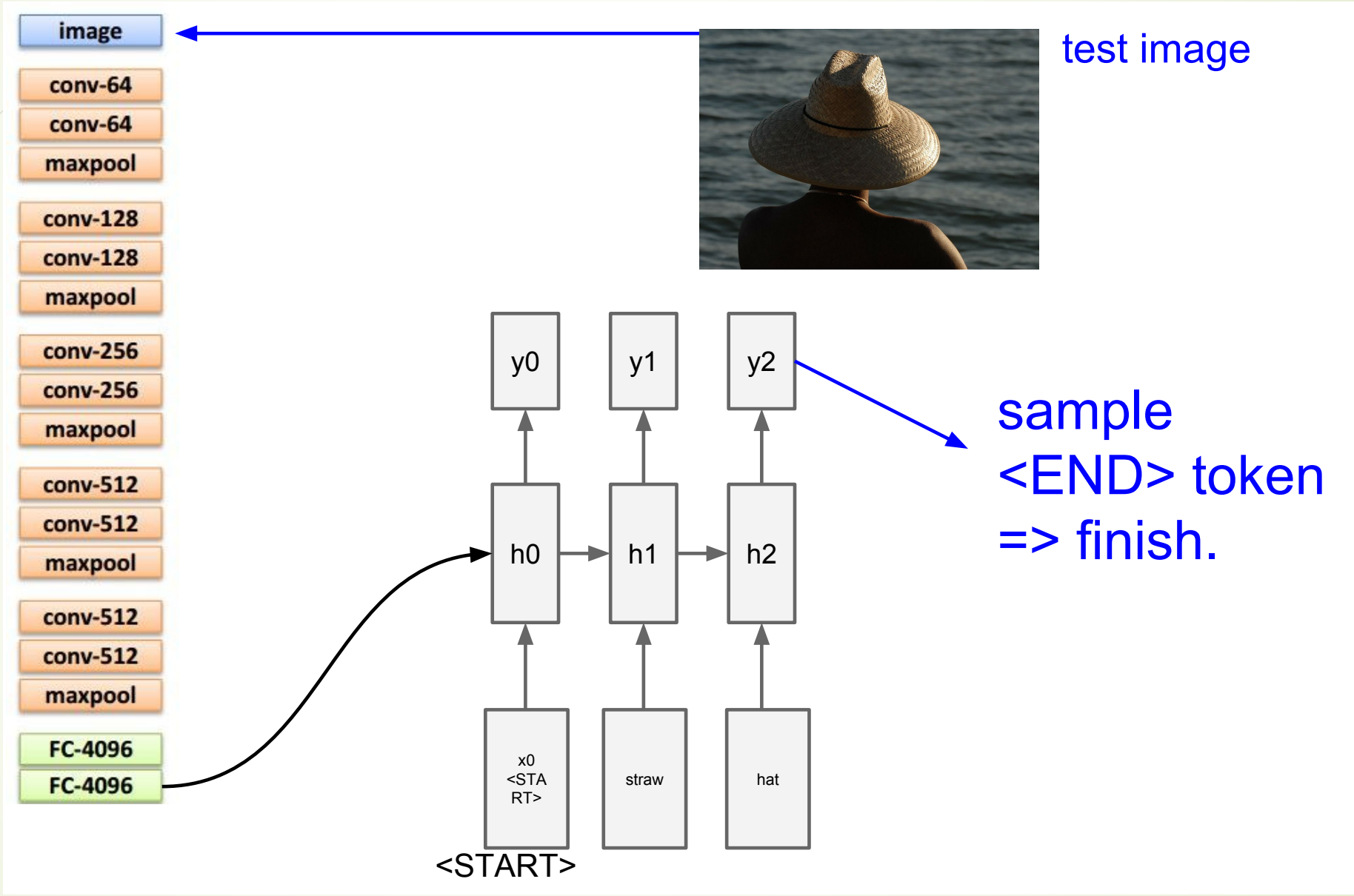


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
h0
h1
h2
x0
<STA
RT>
straw
hat
<START>
sample
<END> token
=> finish.

Image Captioning: Example Results

Captions generated using neuraltalk2
All images are CC0 Public domain:
cat suitcase, cat tree, dog, bear,
surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

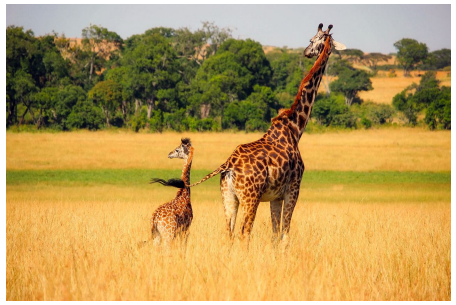
A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court



Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

Long Short Term Memory (LSTM)

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

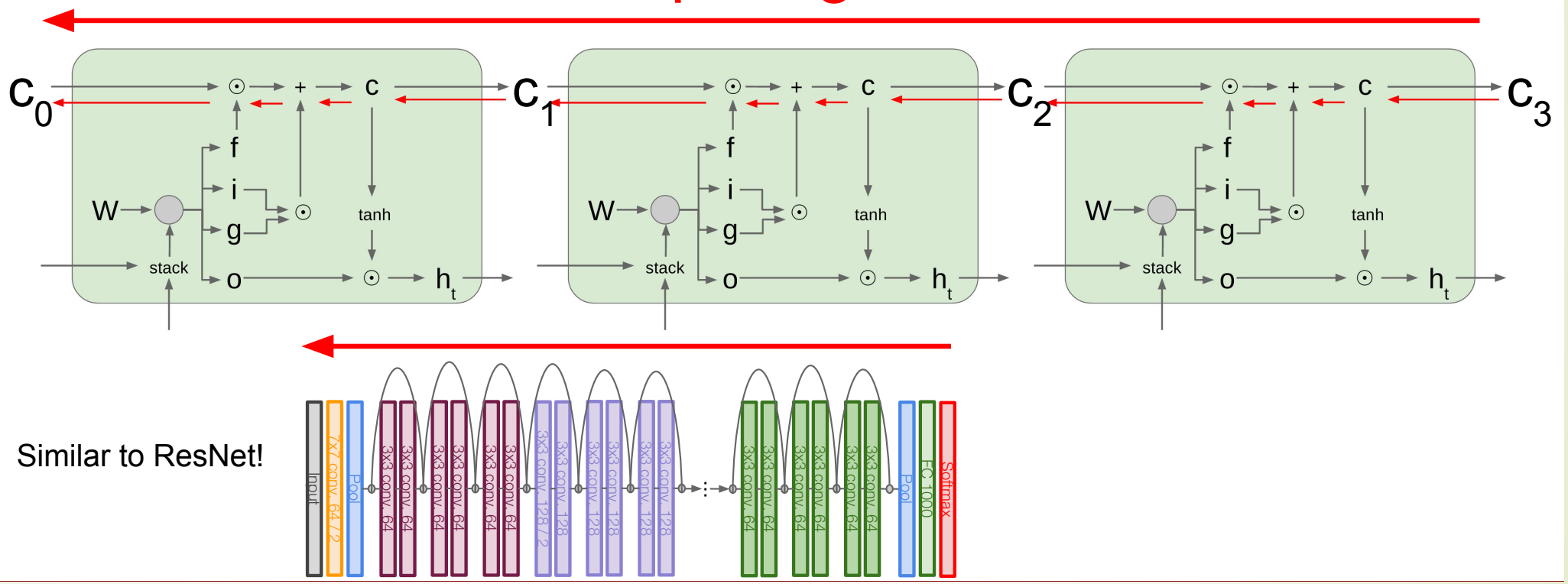
$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!

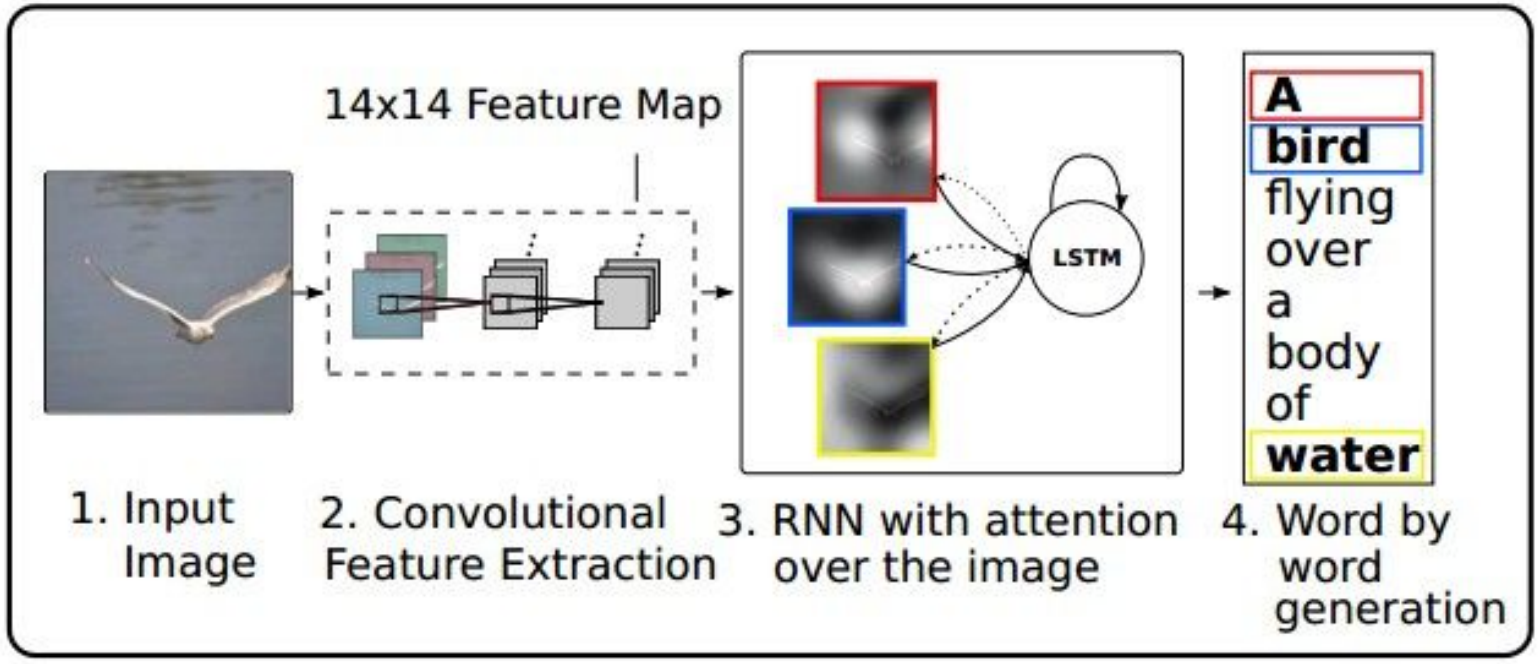


Similar to ResNet!

Like ResNet, LSTM can go deep without the vanishing gradient issue in RNN!

Image Captioning with Attention

RNN focuses its attention at a different spatial location when generating each word



Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Supervised Learning

- **Data**: (x, y)
  x is input, y is output/response (label)
- **Goal**: Learn a *function* to map x -> y
- **Examples**:
  - Classification,
  - regression,
  - object detection,
  - semantic segmentation,
  - image captioning, etc.

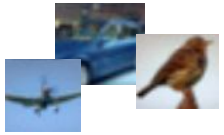
Cat

Unsupervised Learning

- **Data**: x
  Just input data, no output labels!
- **Goal**: Learn some underlying hidden *structure* of the data
- **Examples**:
  - Clustering,
  - dimensionality reduction (manifold learning),
  - feature learning,
  - density estimation,
  - Generating samples, etc.

Generative Models

Given training data, generate new samples from same distribution



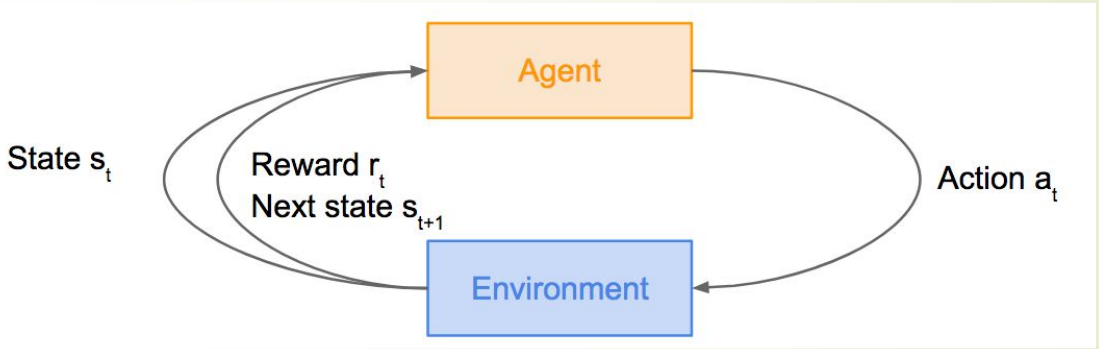
Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

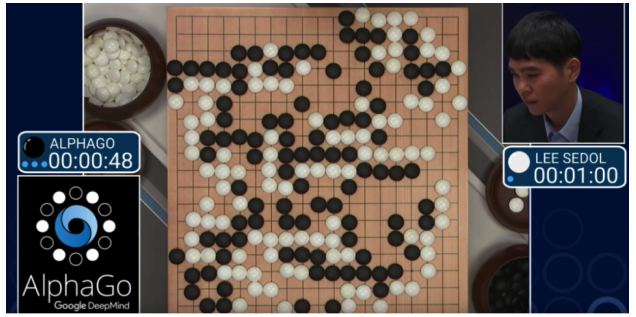
Reinforcement Learning

- Problems involving an **agent**
- interacting with an **environment**,
- which provides numeric **reward** signals
- **Goal**:
  - Learn how to take actions in order to maximize reward

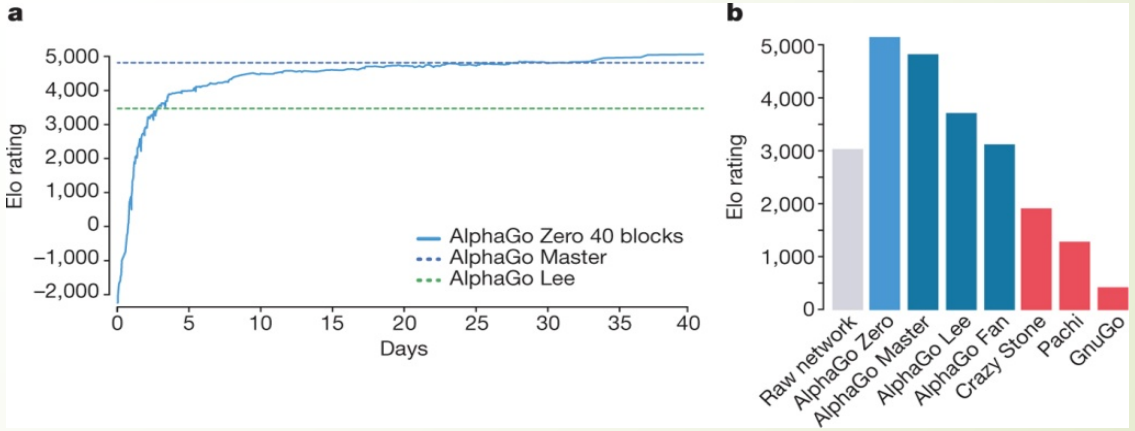


Playing games against human champions

Deep Blue in 1997

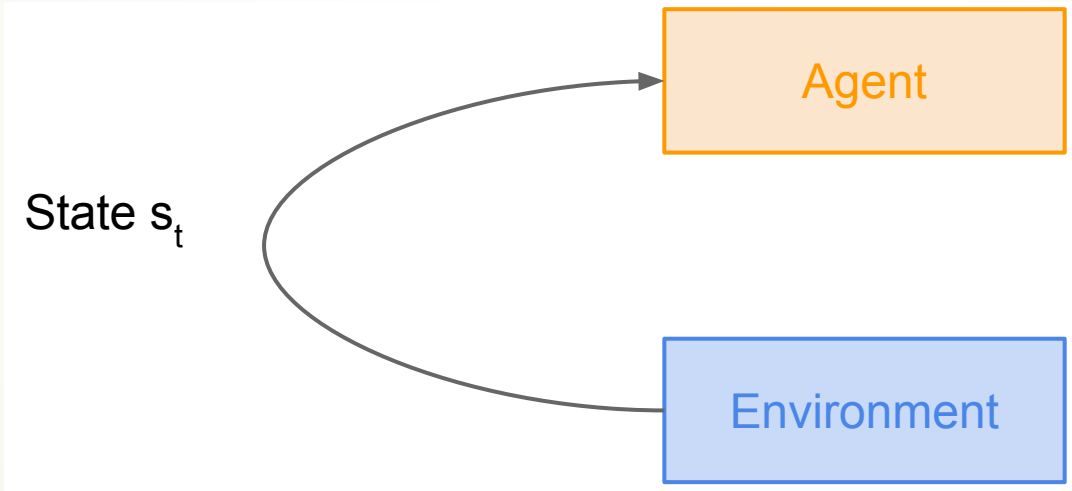


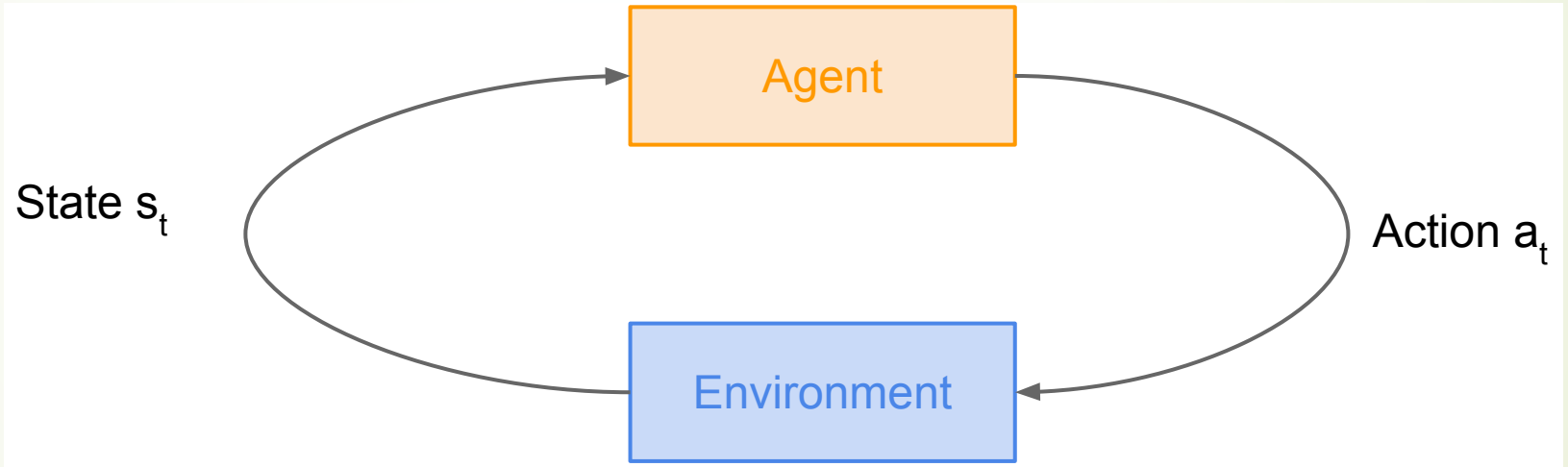
AlphaGo "LEE" 2016

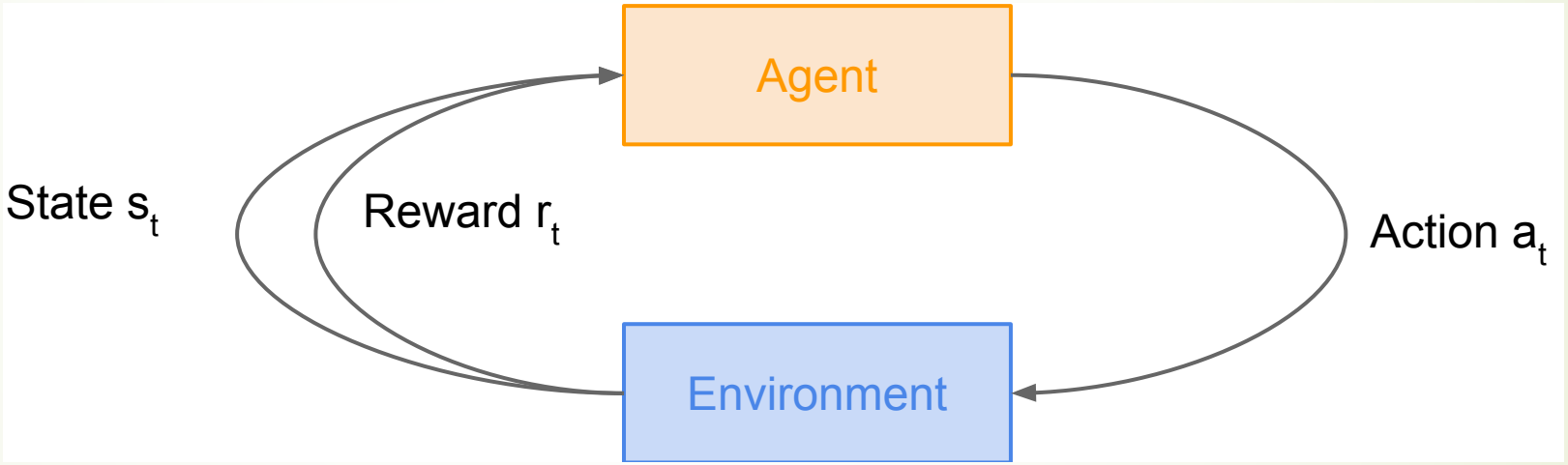


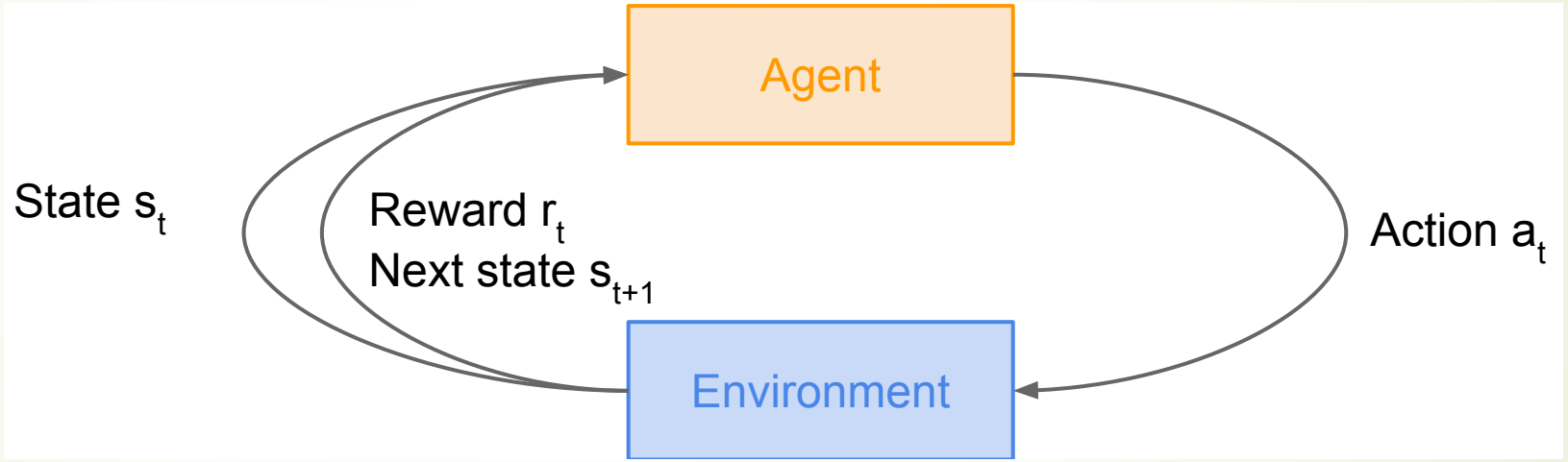
AlphaGo "ZERO" D Silver *et al. Nature* **550,** 354–359 (2017) doi:10.1038/nature24270


Agent
Environment


Agent
State s_t
Environment


Agent
State s_t
Action a_t
Environment


Agent
State s_t
Reward r_t
Action a_t
Environment

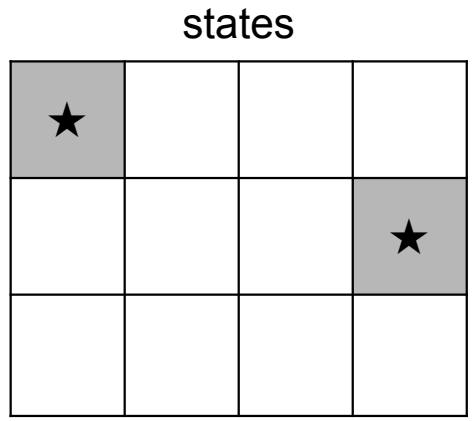

Agent
State s_t
Reward r_t
Next state s_{t+1}
Action a_t
Environment

A simple MDP: Grid World

actions = {

1. right
2. left
3. up
4. down

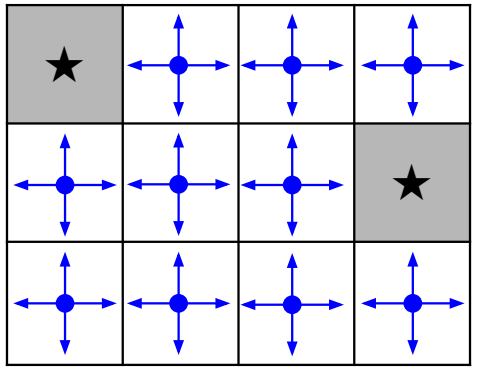
}



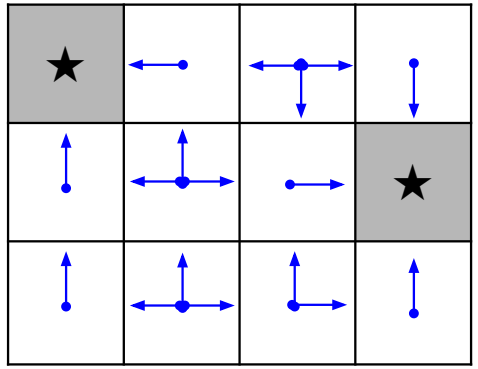
Set a negative “reward” for each transition (e.g. r = -1)

Objective: reach one of terminal states (greyed out) in least number of actions

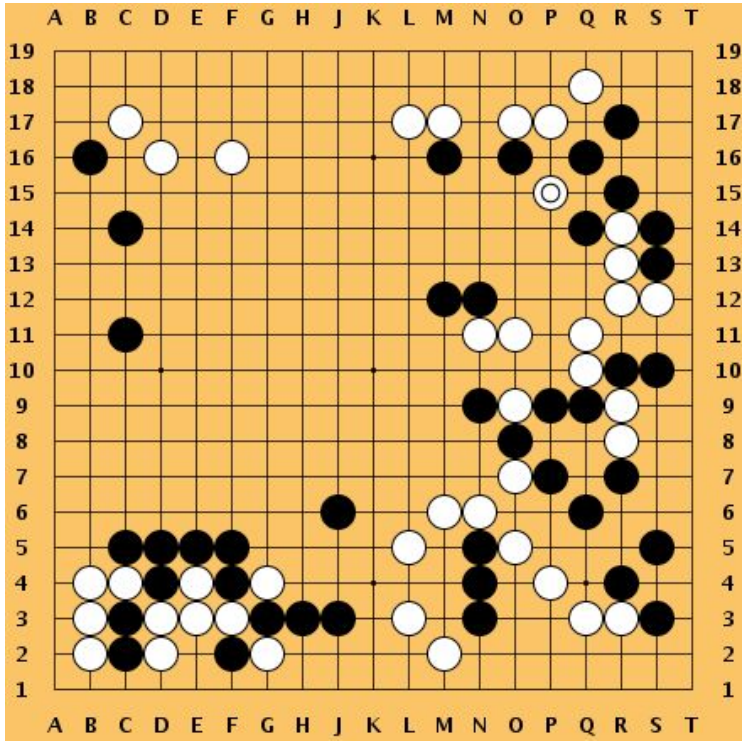
A simple MDP: Grid World



Random Policy



Optimal Policy



Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

Mathematical Formulation of Reinforcement Learning

- **Markov property**: Current state completely characterizes the state of the world

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

- At time step *t=0*, environment samples initial state $s_0 \sim p(s_0)$
- Then, for *t*=0 until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(. \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. \mid s_t; a_t)$
  - Agent receives reward r_t and next state s_{t+1}
- A policy pi is a function from S to A that specifies what action to take in each state
- **Objective**: find policy that maximizes the cumulated discounted reward

Basic Reinforcement Learning

- Discounted future reward – $R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \ldots + \gamma^{n-t} r_n$
- Define Q function - $Q(s_t, a_t) = max\ R_{t+1}$
- Bellman equation - $Q(s, a) = r\ + \gamma max_{a'} Q(s', a')$
- If Q solved or known, best policy is – $\pi(s)\ = argmax_a\ Q(s, a)$
- *Q(s, a)* is representing **the maximum discounted future reward when we perform action** a **in state** s**, and continue optimally from that point on.** *Q* stands for quality of a certain action in a given state.
- In deep reinforcement learning, *Q* is represented by a neural network, as well as the policy map *pi*.

Online resources

Stanford's Convolutional Neural Networks course.
http://cs231n.github.io/
UCL's Reinforcement Learning course.
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
For practicing Reinforcement Learning.
https://gym.openai.com/

Thank you!