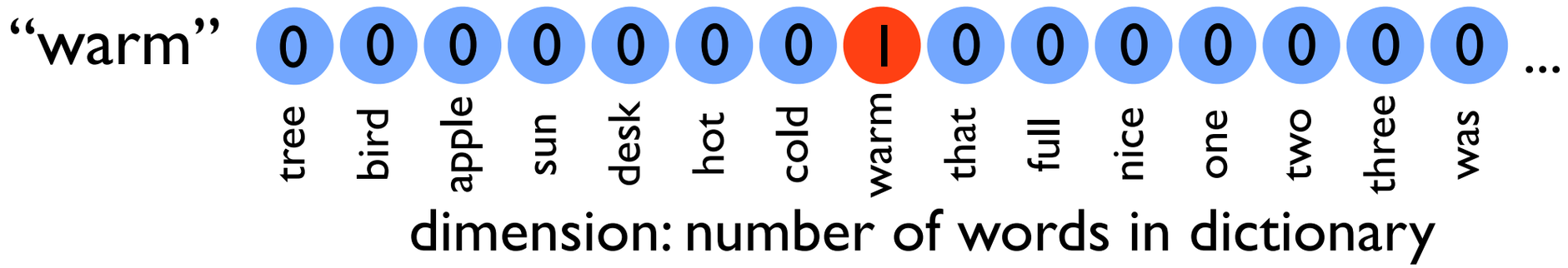
Machine Learning for Physicists
Lecture 9

Summer 2017

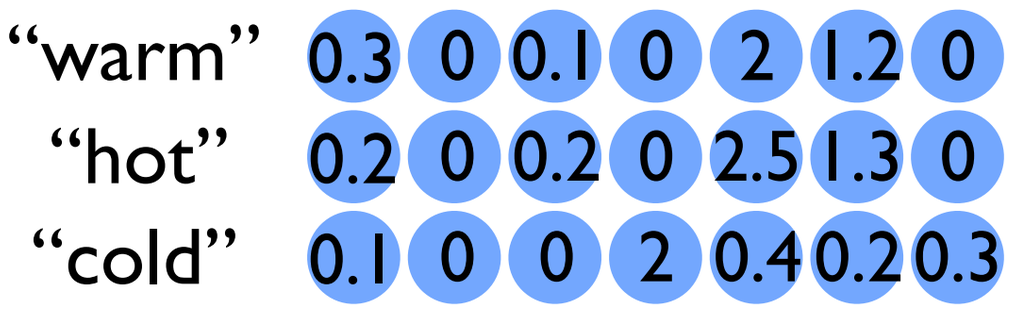
University of Erlangen-Nuremberg
Florian Marquardt

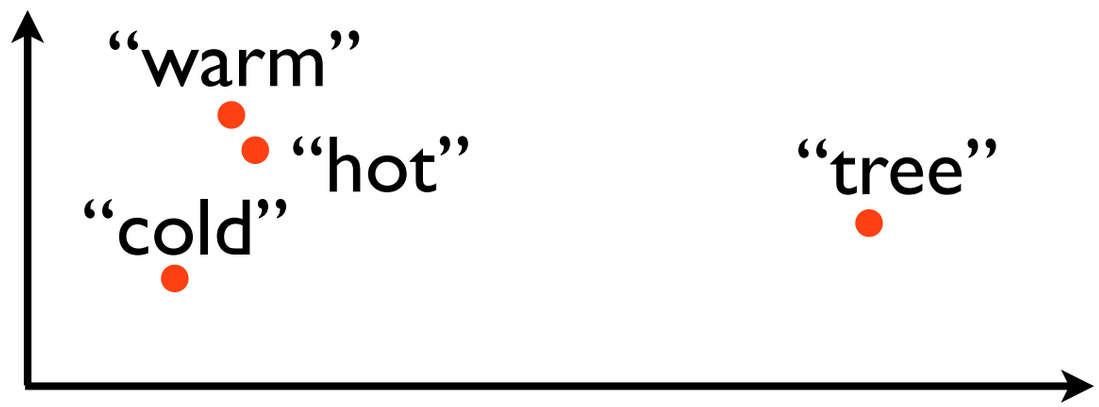
Word vectors

simple one-hot encoding of words needs large vectors (and they do not carry any special meaning):



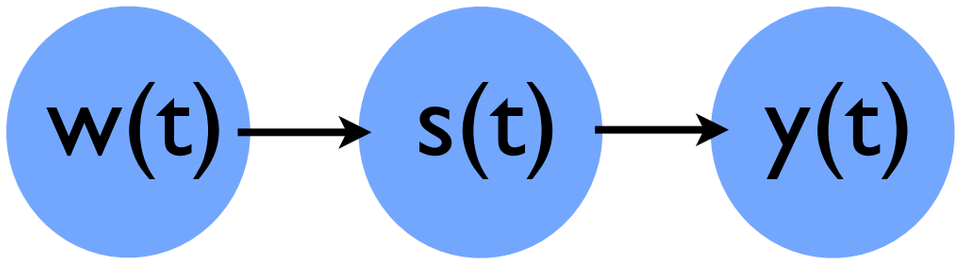
word2vec – reduction to vectors in much lower dimension, where similar words lie closer together:





Word vectors: recurrent net for training

Mikolov, Yih, Zweig 2013



input word at time t (one-hot; dimension D= size of dictionary)

dim. N

predicted next word (probabilities for each word in vocab.; dimension D)

NxD NxN

$$\mathbf{s}(t) = f\left(\mathbf{U}\mathbf{w}(t) + \mathbf{W}\mathbf{s}(t-1)\right)$$

DxN

$$\mathbf{y}(t) = g\left(\mathbf{V}\mathbf{s}(t)\right),$$

where U matrix (NxD) contains word vectors!

$$f(z) = \frac{1}{1+e^{-z}}, \quad g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}.$$

sigmoid SOFTMAX

Word vectors: how to train them

Predicting the probability of any word in the dictionary, given the context words (most recent word): very expensive!

Alternative:

Noise-contrastive estimation: provide a few noisy (wrong) examples, and train the model to predict that they are fake (but that the true one is correct)!

Word vectors: how to train them

Two approaches:

"continuous bag of words" Context words ⟶ word

"skip-gram" word ⟶ context words

Example dataset:

the quick brown fox jumped over the lazy dog

word	context words (here: just surrounding words)
quick	**the, brown**
over	**jumped, the**
lazy	**the, dog**
...	**...**

Word vectors: how to train them

Model tries to predict:

$P_\theta(w, h)$ prob. that w is the correct word, given the context word h

parameters of the model, i.e. weights, biases, and entries of embedding vectors

$$P_\theta(w, h) = \sigma(W_{jk} e_k(h) + b_j)$$

j: index for word w in dictionary
k: index in embedding vector [Einstein sum]
e(h): embedding vector for word h
W,b: weights, biases

At each time-step: go down the gradient of

$$C^{(t)} = \ln P_\theta(w_t, h) + \sum_{\tilde{w}} \ln(1 - P_\theta(\tilde{w}, h))$$

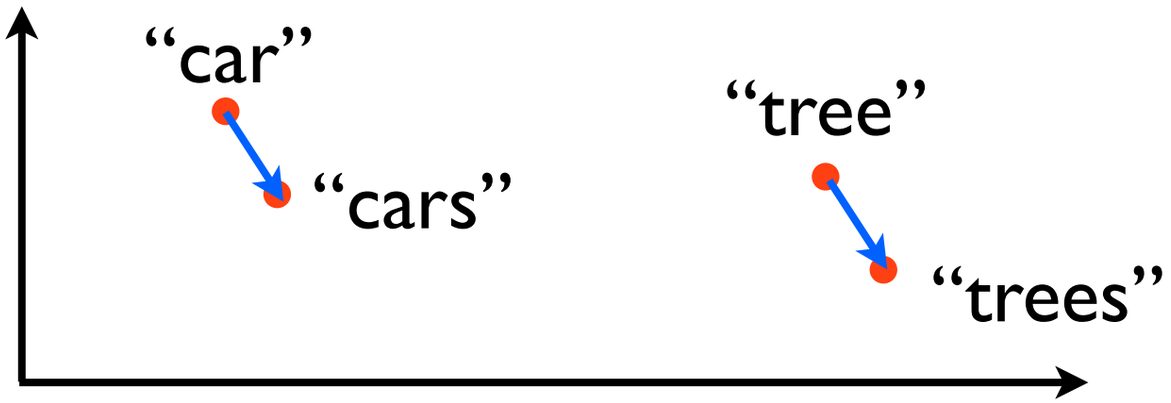
noisy examples

Word vectors encode meaning

Mikolov, Yih, Zweig 2013

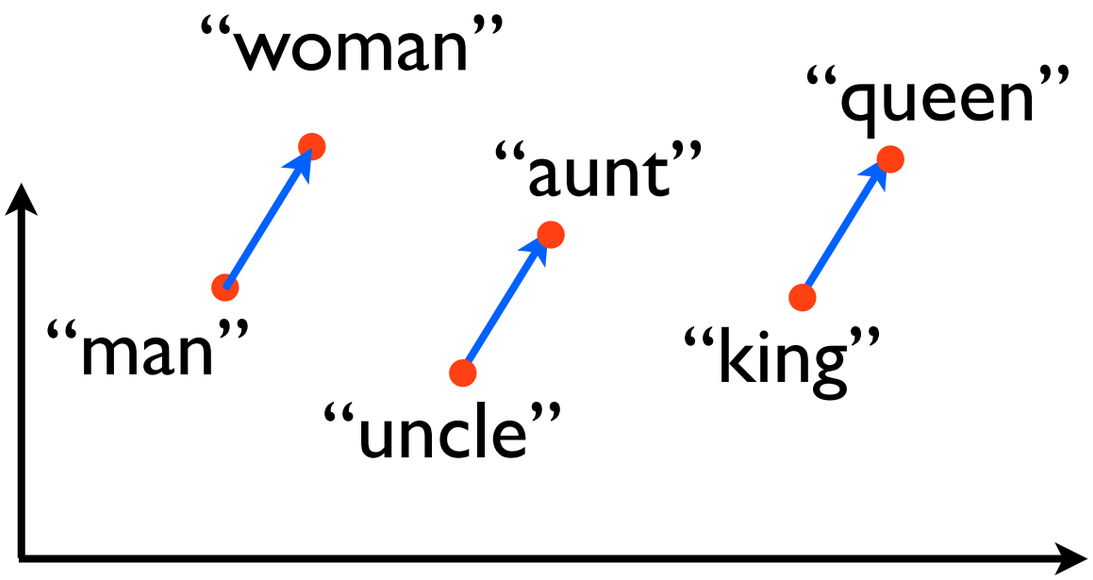
car-cars ~ tree-trees

(subtracting the word vectors on each side yields approx. identical vectors)



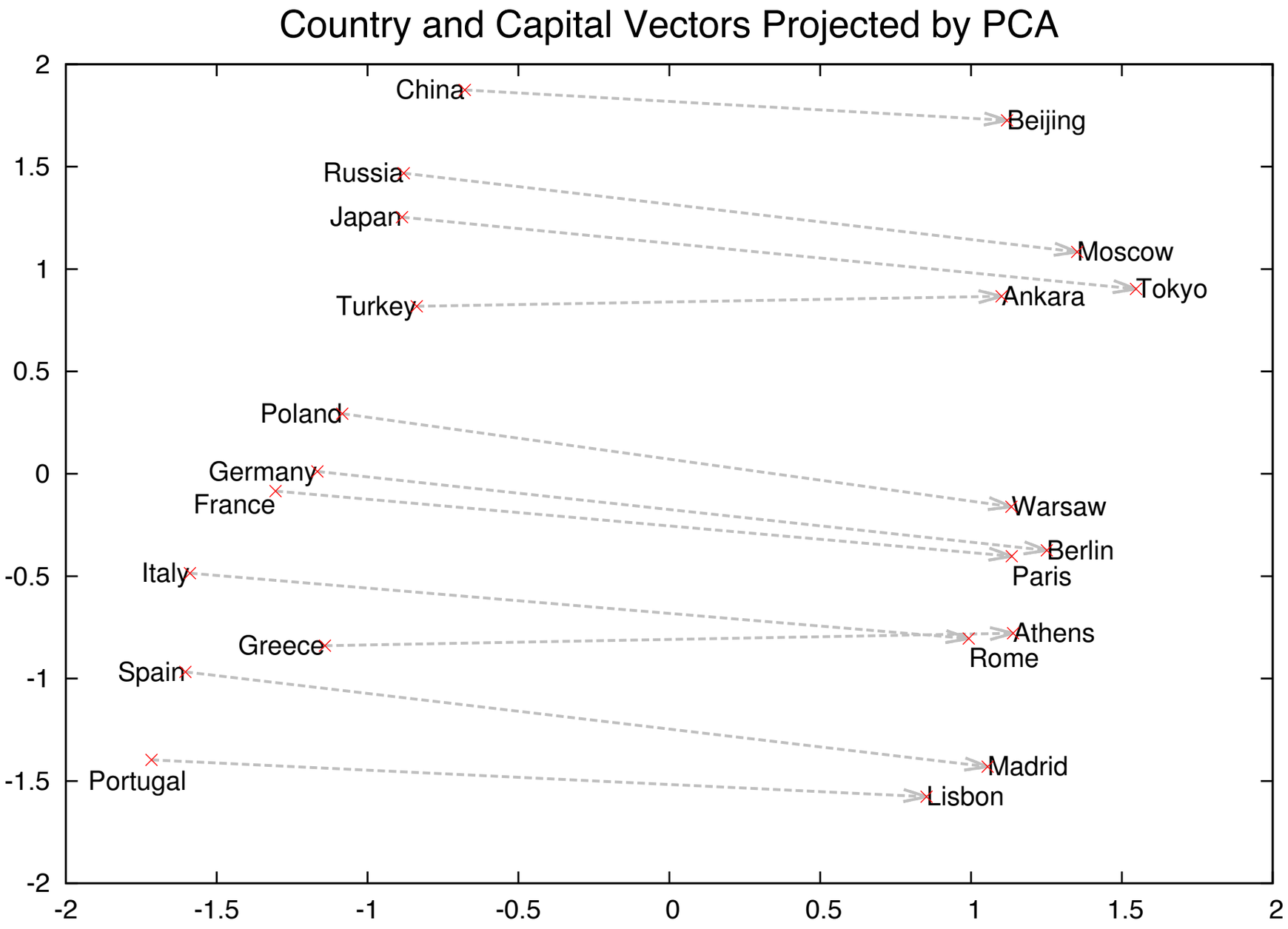
Word vectors encode meaning

Mikolov, Yih, Zweig 2013



Word vectors encode meaning

Mikolov et al. 2013 ”Distributed Representations of Words and Phrases and their Compositionality”



Word vectors in keras

Layer for mapping word indices (integer numbers representing position in a dictionary) to word vectors (of length EMBEDDING_DIM), for input sequences of some given length

```
embedding_layer = Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            input_length=MAX_SEQUENCE_LENGTH)
```

Helper routines for converting actual text into a sequence of word indices. See especially:

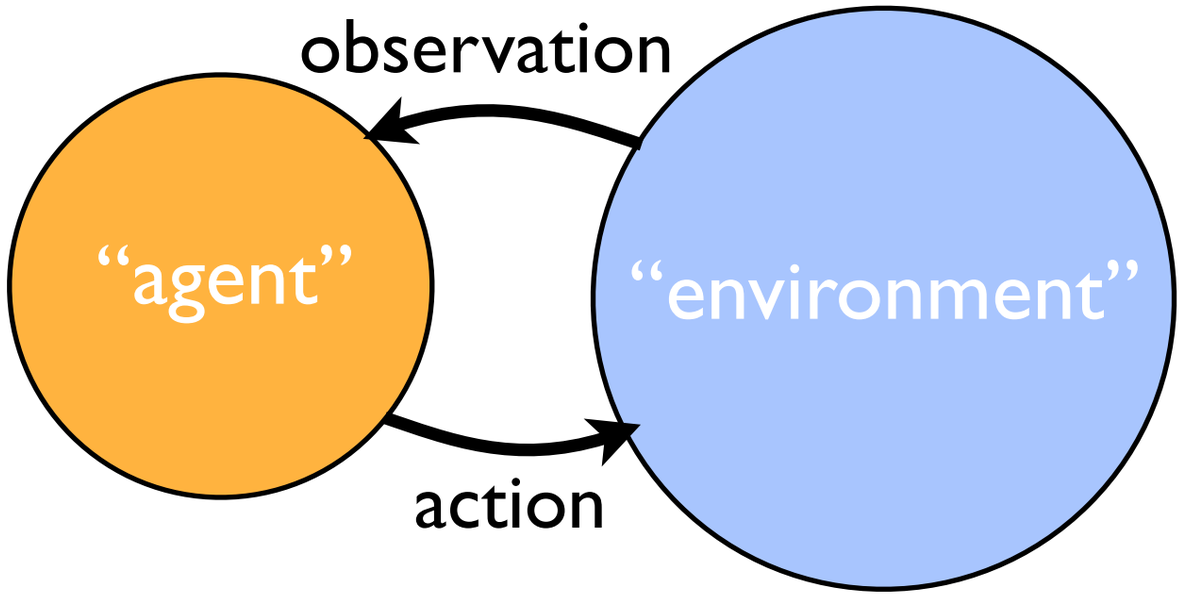
function/class	Keras documentation
`Tokenizer`	Text Preprocessing
`pad_sequences`	Sequence Preprocessing
`(and others)`	

Search for "GloVe word embeddings": 800 MB database pre-trained on a 2014 dump of the English Wikipedia, encoding 400k words in 100-dimensional vectors

Reinforcement Learning

Image: Wikipedia

Reinforcement learning



fully observed vs. partially observed "state" of the environment

Self-driving cars, robotics:
Observe immediate environment & move
Games:
Observe board & place stone
Observe video screen & move player

Challenge: the "correct" action is not known!
Therefore: no supervised learning!

Reward will be rare (or decided only at end)

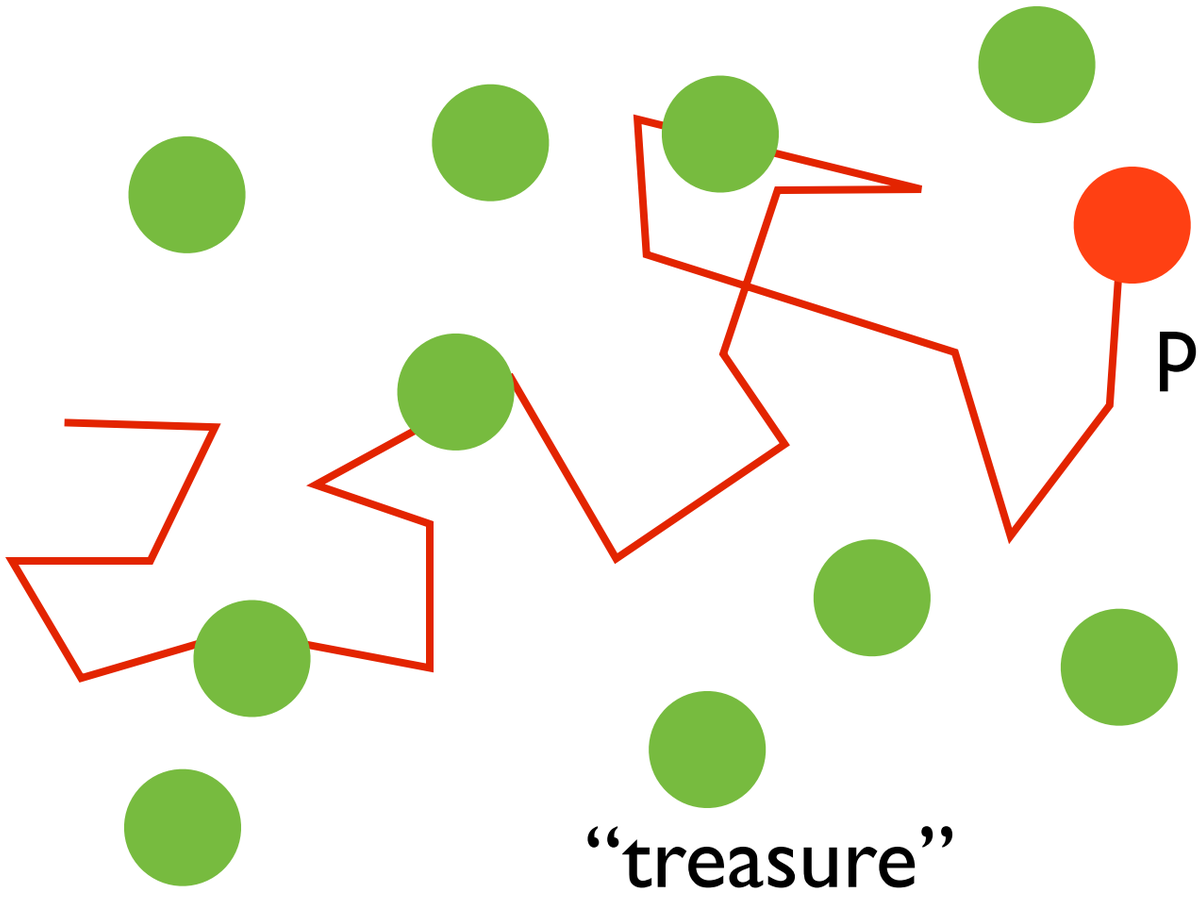
Reinforcement learning

Use **reinforcement learning**:

Training a network to produce actions based on rare rewards (instead of being told the 'correct' action!)

Challenge: We could use the final reward to define a cost function, but we cannot know how the environment reacts to a proposed change of the actions that were taken!

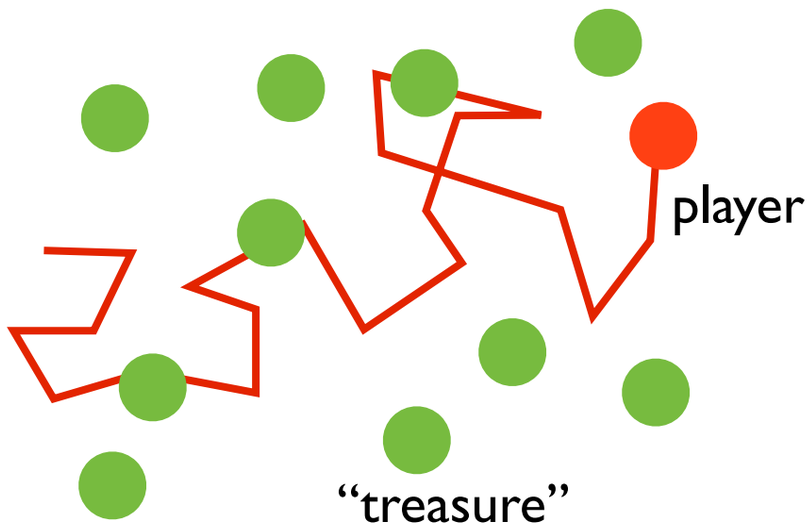
(unless we have a model of the environment)



"State"=full map
"Action"=move
Reward e.g. based on how many "treasures" were collected

Policy Gradient

=REINFORCE (Williams 1992): The simplest model-free general reinforcement learning technique



Basic idea: Use probabilistic action choice. If the reward at the end turns out to be high, make **all** the actions in this sequence **more likely** (otherwise do the opposite)

This will also sometimes reinforce 'bad' actions, but since they occur more likely in trajectories with low reward, the net effect will still be to suppress them!

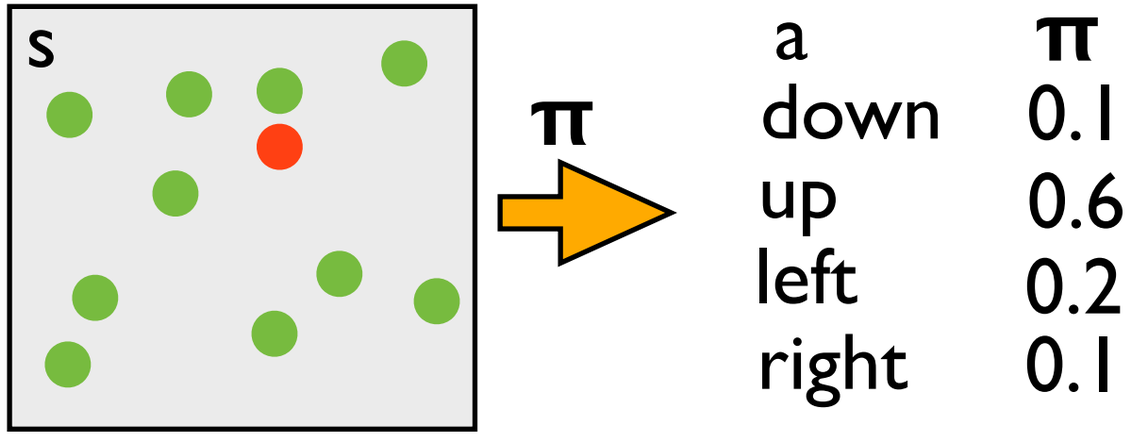
Policy Gradient

Probabilistic policy:

Probability to take action a, given the current state s

$$\pi_\theta(a|s)$$

parameters of the network



Environment: makes (possibly stochastic) transition to a new state s', and possibly gives a reward r

Transition function $P(s'|s,a)$

Policy Gradient

Probability for having a certain trajectory of actions and states:

product over time steps

$$P_\theta(\tau) = \Pi_t P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

trajectory: $\tau = (\mathbf{a}, \mathbf{s})$
$\mathbf{a} = a_0, a_1, a_2, \ldots$
$\mathbf{s} = s_1, s_2, \ldots$ (state 0 is fixed)

Expected overall reward: sum over all trajectories

$$\bar{R} = E[R] = \sum_\tau P_\theta(\tau)R(\tau)$$

— reward for this sequence (sum over individual rewards r for all times)

sum over all actions at all times
and over all states at all times >0

$$\sum_\tau \cdots = \sum_{a_0,a_1,a_2,\ldots,s_1,s_2,\ldots} \cdots$$

Try to maximize expected reward by changing parameters of policy:

$$\frac{\partial \bar{R}}{\partial \theta} = ?$$

Policy Gradient

$$\frac{\partial \bar{R}}{\partial \theta} = \sum_t \sum_\tau R(\tau) \underbrace{\frac{\partial \pi_\theta(a_t|s_t)}{\partial \theta} \frac{1}{\pi_\theta(a_t|s_t)}}_{\frac{\partial \ln \pi_\theta(a_t|s_t)}{\partial \theta}} \Pi_{t'} P(s_{t'+1}|s_{t'}, a_{t'}) \pi_\theta(a_{t'}|s_{t'})$$

Main formula of policy gradient method:

$$\frac{\partial \bar{R}}{\partial \theta} = \sum_t E[R \frac{\partial \ln \pi_\theta(a_t|s_t)}{\partial \theta}]$$

Stochastic gradient descent:

$$\Delta\theta = \eta \frac{\partial \bar{R}}{\partial \theta}$$

where $E[\ldots]$ is approximated via the value for one trajectory (or a batch)

Policy Gradient

$$\frac{\partial \bar{R}}{\partial \theta} = \sum_t E[R \frac{\partial \ln \pi_\theta(a_t|s_t)}{\partial \theta}]$$

Increase the probability of all action choices in the given sequence, depending on size of reward R.
Even if R>0 always, due to normalization of probabilities this will tend to suppress the action choices in sequences with lower-than-average rewards.

Abbreviation:

$$G_k = \frac{\partial \ln P_\theta(\tau)}{\partial \theta_k} = \sum_t \frac{\partial \ln \pi_\theta(a_t|s_t)}{\partial \theta_k}$$

$$\frac{\partial \bar{R}}{\partial \theta_k} = E[RG_k]$$

Policy Gradient: reward baseline

Challenge: fluctuations of estimate for reward gradient can be huge. Things improve if one subtracts a constant baseline from the reward.

$$\frac{\partial \bar{R}}{\partial \theta} = \sum_t E[(R - b)\frac{\partial \ln \pi_\theta(a_t|s_t)}{\partial \theta}]$$
$$= E[(R-b)G]$$

This is the same as before. Proof:

$$E[G_k] = \sum_\tau P_\theta(\tau)\frac{\partial \ln P_\theta(\tau)}{\partial \theta_k} = \frac{\partial}{\partial \theta_k}\sum_\tau P_\theta(\tau) = 0$$

However, the variance of the fluctuating random variable (R-b)G is different, and can be smaller (depending on the value of b)!

Optimal baseline

$$X_k = (R - b_k)G_k$$

$$\mathrm{Var}[X_k] = E[X_k^2] - E[X_k]^2 = \min$$

$$\frac{\partial \mathrm{Var}[X_k]}{\partial b_k} = 0$$

$$b_k = \frac{E[G_k^2 R]}{E[G_k^2]}$$

$$G_k = \frac{\partial \ln P_\theta(\tau)}{\partial \theta_k}$$

$$\Delta\theta_k = -\eta E[G_k(R - b_k)]$$