Computational Cognitive Neuroscience
Theoretical background

Marcel van Gerven

Introduction to three important computational paradigms

- Bayesian modeling
- Artifical neural networks
- Reinforcement learning

- You will work on these three topics during the practical

Part I: Bayesian Modeling

Bayesian modeling

Bayesian statistics is concerned with updating our beliefs in the face of uncertainty given observed data.

We can use these updated beliefs to solve three fundamental inference problems:

1. Parameter estimation

2. Prediction

3. Model comparison

Relevance for cognitive neuroscience

Provides a normative account of what the brain is doing.

That is, the brain essentially tries to infer the causes of its sensory input in order to make optimal decisions. This is known as the Bayesian brain hypothesis.

Provides the ingredients for building computational models that can deal with uncertainty.

For example, when modeling how visual input leads to neuronal activity, we need to take into account sensory noise. Model comparison allows us to differentiate between different competing models.

Provides a principled approach to data analysis.

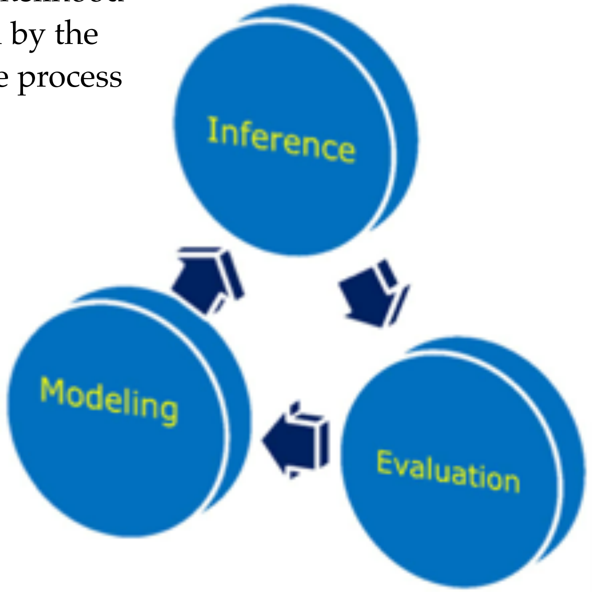
For example, when we try to estimate how brain networks are anatomically connected using diffusion imaging (a magnetic resonance imaging technique), we need to take into account that the observed data is affected by various sources of noise.

Allows us to make optimal predictions.

For example in the context of clinical neuroscience or in brain-computer interface research.

Bayesian modeling

Infer parameters from the joint likelihood induced by the generative process



Devise a generative process for the data. I.e. a recipe for how the observed world could be generated

Evaluate model parameters by how well the model is able to predict unobserved quantities or account for observed structures

Bayesian modeling

- Models have parameters some of which (if not all) are unknown, e.g. $\boldsymbol{\theta}$.
- In statistical modelling we are interested in inferring (e.g. estimating) unknown parameters from data → inference.
- Parameter estimation needs be done in a formal way. In other words we ask ourselves the question: what are the best values for $\boldsymbol{\theta}$ such that a proposed model best describes the observed data?
- Should we only look for a single estimate for $\boldsymbol{\theta}$?
- No! There may be many values $\boldsymbol{\theta}$ (often not very different from each other) which may equally well describe the data → uncertainty

Bayes' rule

A model describes data D that one could observe from a system

- If we use the mathematics of probability theory to express all forms of uncertainty associated with our model …
- … then inverse probability allows us to infer unknown quantities **θ**, adapt our models, make predictions and learn from data.

- Inverse probability (i.e. statistics) is given by Bayes' rule:

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}$$

- Bayes' rule tells us how to do inference about hypotheses from data.
- Learning and prediction can be seen as forms of inference.

Bayes' rule

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}$$

where

- p(**θ**|D) denotes the posterior distribution of the parameters given the data;
- p(D|**θ**), also written as L(**θ**) is the likelihood function;
- p(**θ**) is the prior distribution of **θ** which express our beliefs about the parameters, before we see the data;
- p(D) is called the marginal likelihood (evidence) and plays the role of the normalising constant of the density of the posterior distribution

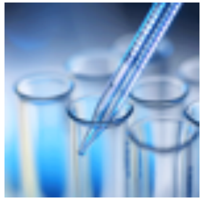
Bayes' rule

- Everything is assigned distributions (prior, posterior)
- we are allowed to incorporate prior information about the parameters . . .
- which is then updated by using the likelihood function . . .
- leading to the **posterior distribution** which tell us everything there is to know about the parameters.

Example

Let's think of a very simple example:

- Suppose we are interested in estimating the probability of success (denoted by θ) for a particular experiment.



Data:

Out of *n* times we repeated the experiment we observed *k* successes.

$\downarrow$ fixed parameter!

$\downarrow$ data!

Bayes' rule gives:

$$p(\theta \mid n, k) = \frac{p(k \mid n, \theta)p(\theta)}{p(k)}$$

Likelihood function

We need to specify the likelihood function:

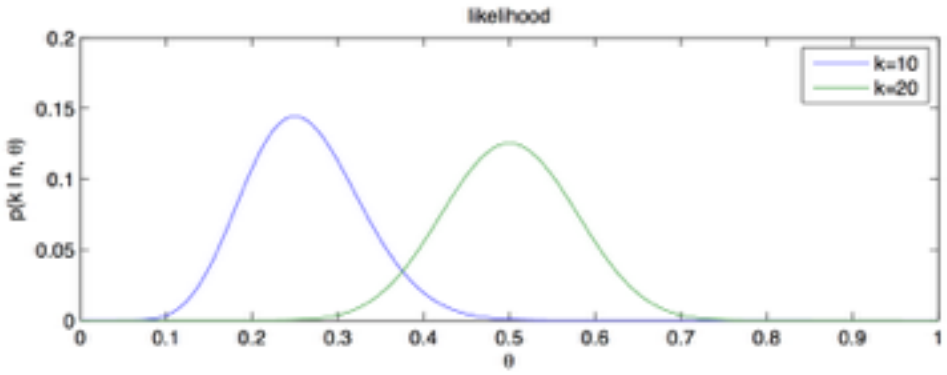
binomial coefficient n!/k!(n-k)!

number of ways to choose k objects out of n objects

$$p(k \mid n, \theta) = \text{Binomial}(k \mid n, \theta) = \binom{n}{k} \theta^k (1-\theta)^{n-k}$$

probability of success

probability of failure



Prior

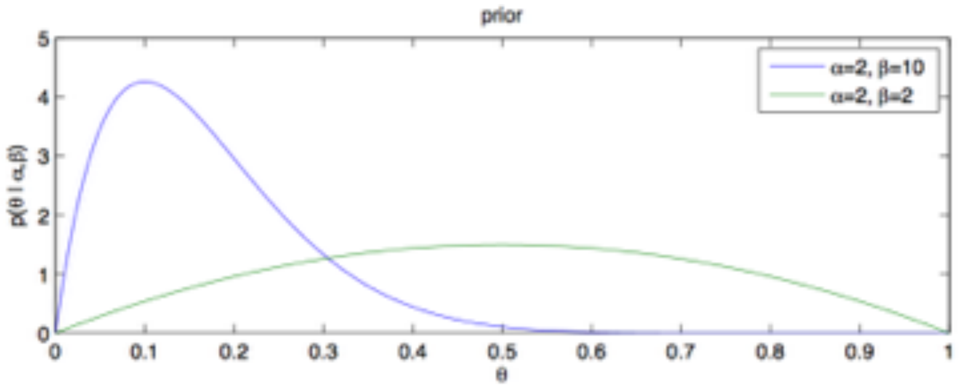
We also need to specify a prior:

pseudocount for number of successes

pseudocount for number of failures

$$p(\theta \mid \alpha, \beta) = \mathrm{Beta}(\theta \mid \alpha, \beta) = \frac{1}{B(\alpha,\beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1}$$

where $B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}\,\mathrm{d}t$ is the Beta function



Prior

We are interested in estimating the probability of success (denoted by θ) for one particular experiment.

Data:

Out of *n* times we repeated the experiment we observed *k* successes.

Specification of the likelihood function:

$$p(k \mid n, \theta) = \text{Binomial}(k \mid n, \theta) = \binom{n}{k} \theta^k (1-\theta)^{n-k}$$

Specification of the prior:

$$p(\theta \mid \alpha, \beta) = \text{Beta}(\theta \mid \alpha, \beta) = \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1}$$

where $B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}\,\mathrm{d}t$ is the Beta function

Bayesian inference

How to compute the posterior of the Beta-Binomial model?

The Beta-Binomial model is one of the few models for which an analytical solution to the posterior is available.

The availability of such an analytical solution depends on the notion of **conjugacy**.

The prior and likelihood are **conjugate** when the posterior takes the same functional form as the prior distribution.

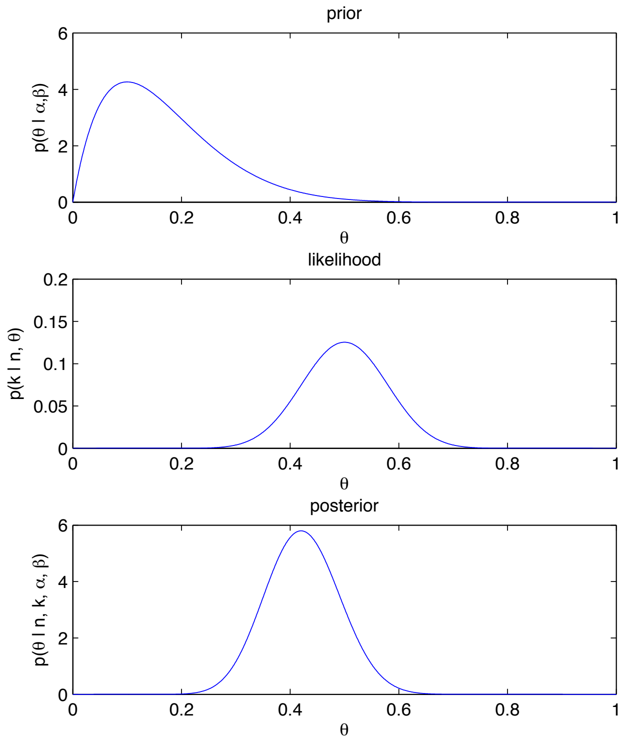
The analytical solution of the posterior is given by:

$$p(\theta \mid k, n, \alpha, \beta) = \frac{1}{B\left(k+\alpha, n-k+\beta\right)} \theta^{k+\alpha-1}(1-\theta)^{n-k+\beta-1}$$

which is equal to a Beta distribution with parameters k+α and n-k+β.

Bayesian inference

Posterior distribution for θ given k = 20, n = 40, α = 2, β = 10



Motivation for Bayesian statistics

When drawing inference within a Bayesian framework,

- the **data** are treated as a **fixed** quantity and
- the **parameters** are treated as **random** variables.

This is exactly opposite from frequentist statistics where

- the data are treated as a **random** quantity
- the parameters are treated as a **fixed** quantity (point estimate)

Bayesian statistics allows us to assign to parameters (and models) probabilities, making the inferential framework

- **intuitive** and
- **straightforward** (at least in principle!)

Point estimation

While, in principle, Bayesian inference is straightforward and intuitive when it comes to computations it **can be tricky to implement**.

An often used simplification is to compute a point estimate of $\boldsymbol{\theta}$ rather than a full posterior distribution.

Consider again Bayes' rule

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}$$

The evidence does not involve the parameters $\boldsymbol{\theta}$, and is given by a single number that ensures that the area under the posterior distribution equals one.

MAP estimation

Therefore, Bayes' rule is also written as

$$p(\boldsymbol{\theta} \mid \mathcal{D}) \propto p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})$$

A reasonable point estimate is the maximizer of this quantity:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \left[p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) \right]$$

This is known as the maximum a posteriori (MAP) estimate

ML estimation

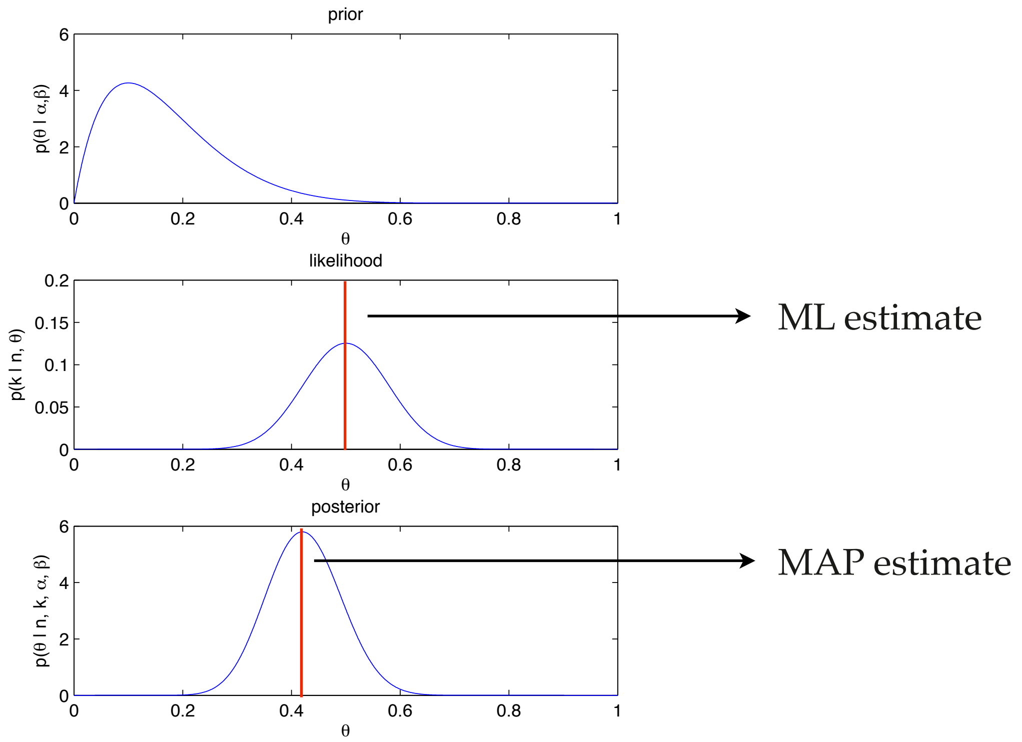
In case we have a flat prior (all values of θ equally likely), the MAP estimate reduces to the maximum likelihood (ML) estimate:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \left[p(\mathcal{D} \mid \boldsymbol{\theta}) \right]$$

These formulations turn a difficult integration problem into an easier optimisation problem.

Comparison

Posterior distribution for θ given $k = 20$, $n = 40$, $\alpha = 2$, $\beta = 10$



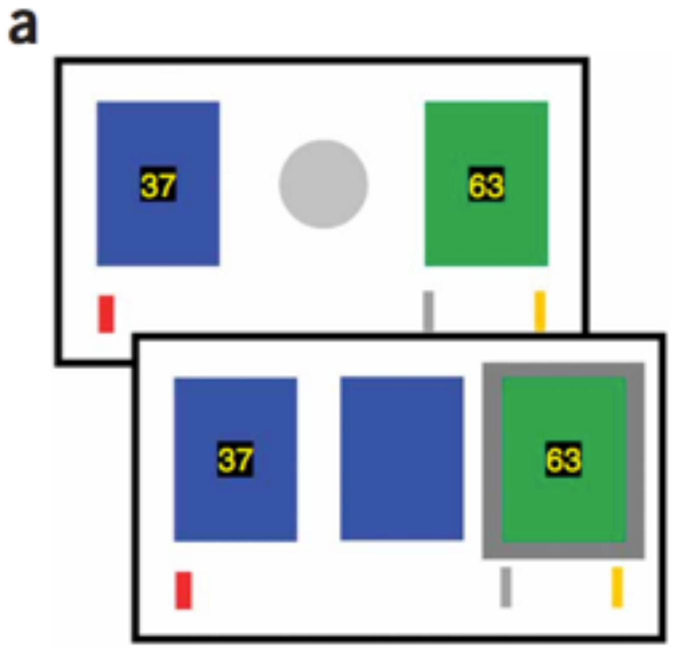
Approximate inference

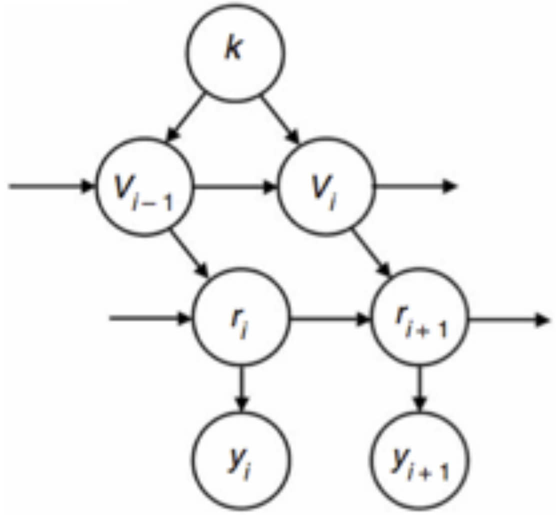
In case there is no analytical solution and we do wish to compute a full posterior, we can use:

- explicit enumeration
- deterministic approximate inference
- stochastic approximate inference

This is a very active research area

Example: Learning about hidden states in the environment





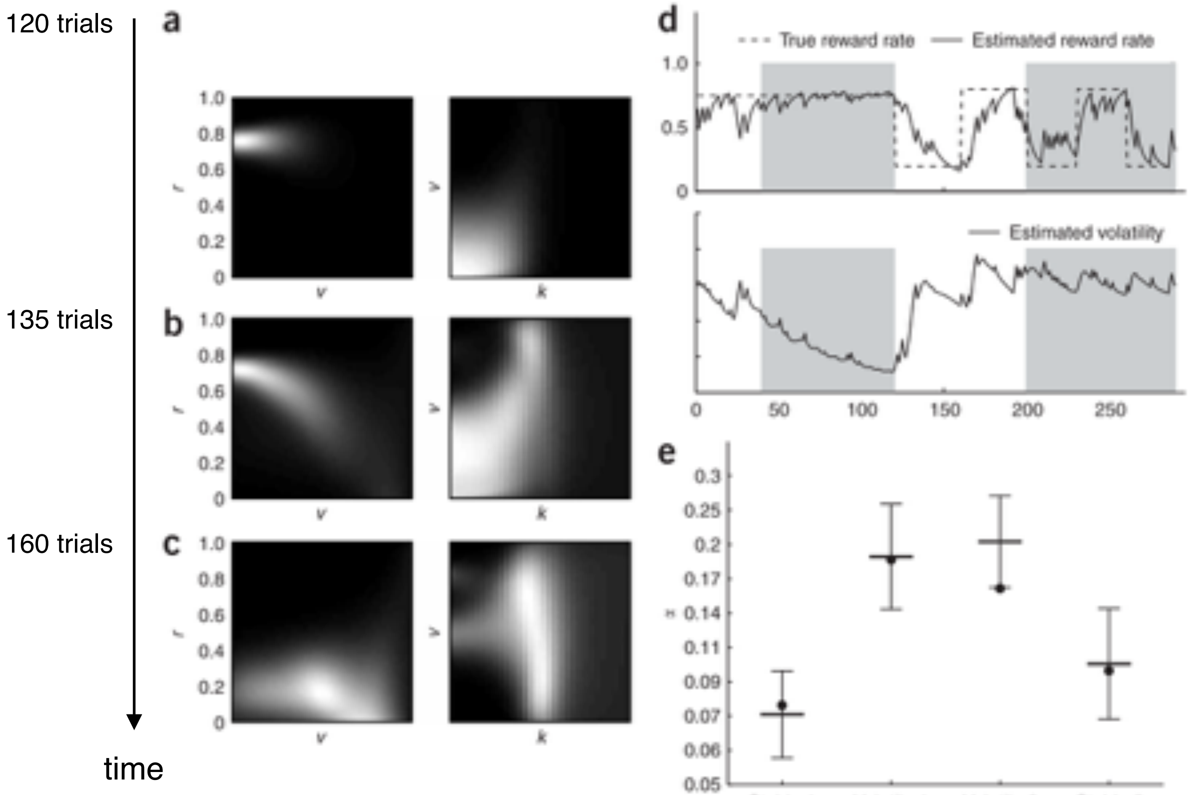
rate of volatility change

volatility

probability color correct

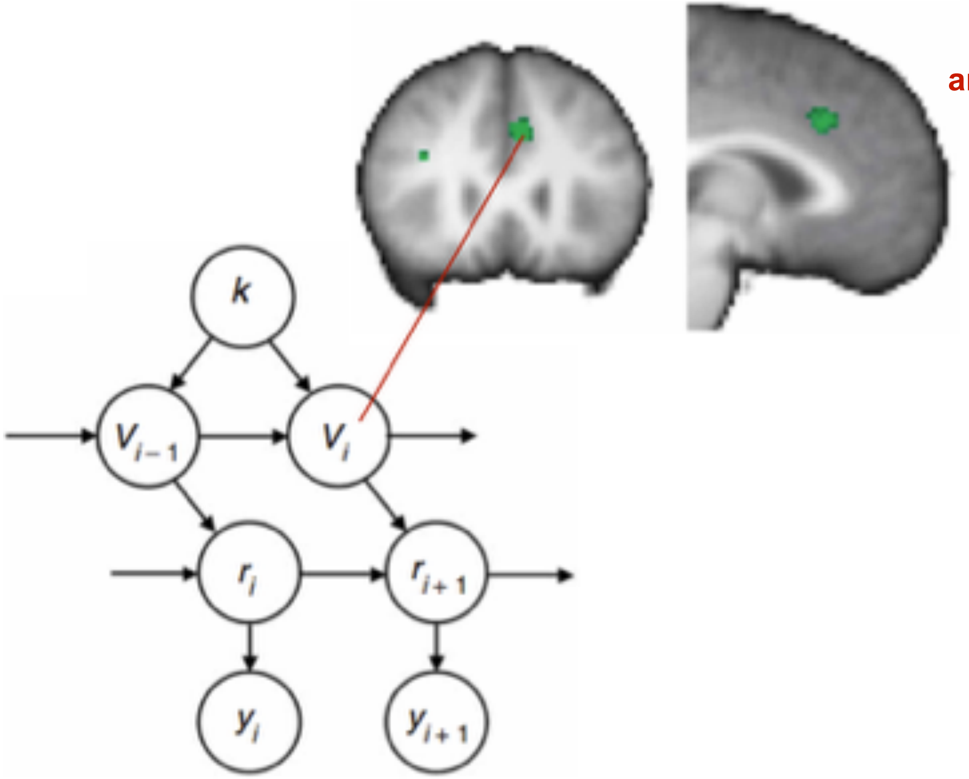
feedback

Example: Learning about hidden states in the environment



Behrens TEJ, Woolrich MW, Walton ME, Rushworth MFS. Learning the value of information in an uncertain world. Nat. Neurosci. 2007 Sep;10(9):1214–21.

Example: Learning about hidden states in the environment



anterior cingulate codes for volatility

Behrens TEJ, Woolrich MW, Walton ME, Rushworth MFS. Learning the value of information in an uncertain world. Nat. Neurosci. 2007 Sep;10(9):1214–21.

Model comparison

Consider again Bayes' rule

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}$$

The evidence (marginal likelihood) implicitly depends on our chosen model and is given by

$$p(\mathcal{D} \mid M) = \int p(\mathcal{D} \mid \boldsymbol{\theta}, M)p(\boldsymbol{\theta} \mid M)\, d\boldsymbol{\theta}$$

It plays an important role in Bayesian statistics and forms the basis for model comparison.

Bayes factor

Consider two alternative models M_1 and M_2.

We can make a relative statement about which model is more likely by computing the **Bayes factor***:*

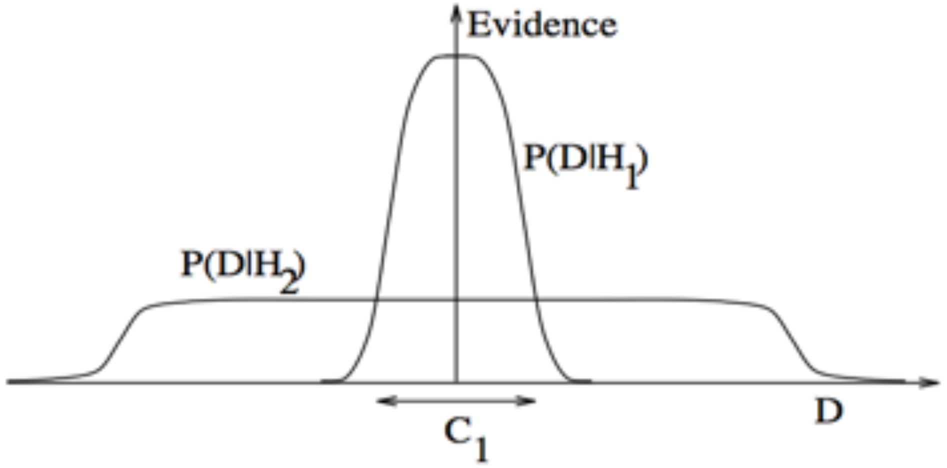
$$K = \frac{p(M_1 \mid \mathcal{D})}{p(M_2 \mid \mathcal{D})} = \frac{p(\mathcal{D})^{-1} p(\mathcal{D} \mid M_1) p(M_1)}{p(\mathcal{D})^{-1} p(\mathcal{D} \mid M_2) p(M_2)} = \frac{p(\mathcal{D} \mid M_1)}{p(\mathcal{D} \mid M_2)}$$

where we made use of the assumption that all models

Bayes factors automatically penalize for model complexity.

Occam’s razor

Occam’s razor follows because simple hypotheses (models) make precise predictions whereas complex hypotheses can make a great variety of predictions. These complex hypotheses must spread their predictive probability P(D | H) more thinly over the data space compared to the simple hypotheses.



After MacKay.

DCM example

DCM EXAMPLE

Making decisions

Inference as such is not of much use when it is not accompanied by decisions. E.g., model comparison entails committing to a hypothesis, predictions entail the selection of an action, et cetera.

This is the realm of decision theory. Define a **loss function**

$$L(\boldsymbol{\theta}, a(\mathbf{x}))$$

which quantifies the loss when $\boldsymbol{\theta}$ parameterizes the true model (state of the world) and a(**x**) is the action performed based on observations **x**.

After MacKay.

Making decisions

Bayesian decision theory states that the optimal decision rule given an observation **x** is the one which minimizes the **posterior expected loss**:

$$\mathbb{E}\left[L(\boldsymbol{\theta}, a(\mathbf{x})) \mid \mathbf{x}\right] = \int L(\boldsymbol{\theta}, a(\mathbf{x})) p(\boldsymbol{\theta} \mid \mathbf{x}) d\boldsymbol{\theta}$$

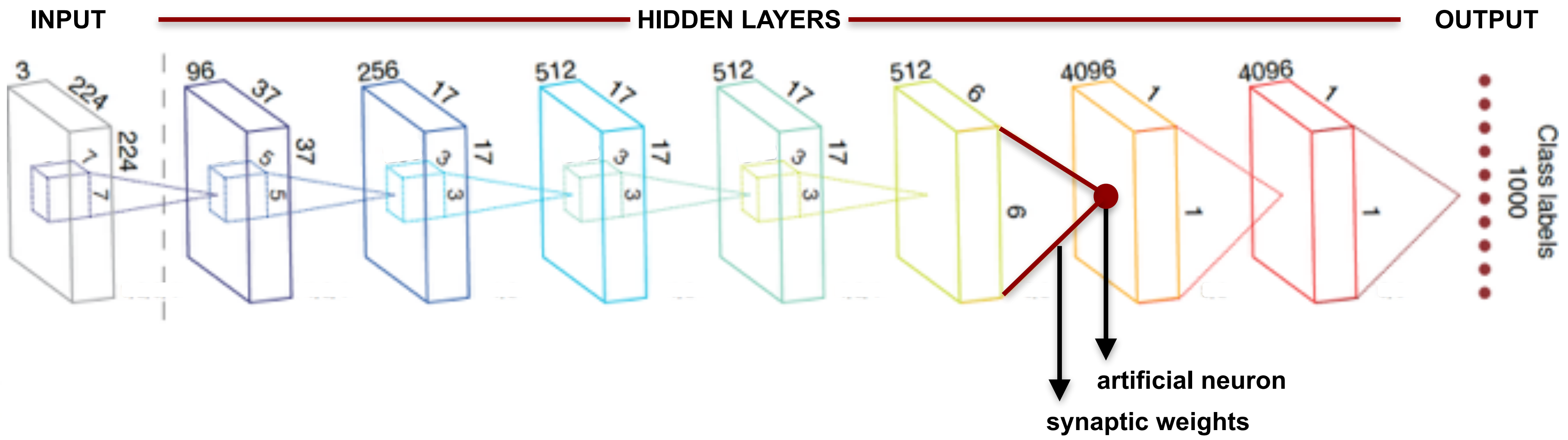
Extension to decision making over time leads to **optimal control theory.**

Optimal control theory lays the groundwork for theories of animal learning and motor control.

After MacKay.

Part II: Artificial Neural Networks

Artificial neural networks



- Computational model consisting of artificial neurons that are organized in terms of layers
- Each layer incorporates particular filtering steps
- Learns a mapping between an input (e.g. image) and an output (e.g. image categories) by adjusting ‘synaptic’ weights between artificial neurons
- Representations in hidden layers capture increasingly complex characteristics of their input

Forward pass

We consider the classical MLP with two layers (one hidden layer)

$$\mathbf{y} = \mathbf{f}(\mathbf{W}^2\mathbf{f}(\mathbf{W}^1\mathbf{x}))$$

where **f** is the activation function applied to all input elements.

First layer weight matrix and input vector (*J* inputs and *I* hidden units):

$$\boldsymbol{W}^1 = \begin{bmatrix} \boxed{w^1_{11} \quad \cdots \quad w^1_{1J}} \\ \vdots \quad \ddots \quad \vdots \\ w^1_{I1} \quad \cdots \quad w^1_{IJ} \end{bmatrix} \quad \boldsymbol{x} = \begin{bmatrix} \boxed{\begin{matrix} x_1 \\ \vdots \\ x_J \end{matrix}} \end{bmatrix}$$

Input activation (an *I*-dimensional vector):

$$\mathbf{W}^1\mathbf{x} = \begin{bmatrix} w^1_{11}x_1 + w^1_{12}x_2 + \cdots + w^1_{1J}x_J \\ \vdots \\ w^1_{I1}x_1 + w^1_{I2}x_2 + \cdots + w^1_{IJ}x_J \end{bmatrix}$$

Forward pass

We consider the classical MLP with two layers (one hidden layer)

$$\mathbf{y} = \mathbf{f}(\mathbf{W}^2\mathbf{f}(\mathbf{W}^1\mathbf{x}))$$

where **f** is the activation function applied to all input elements.

Output (element-wise application of activation function):

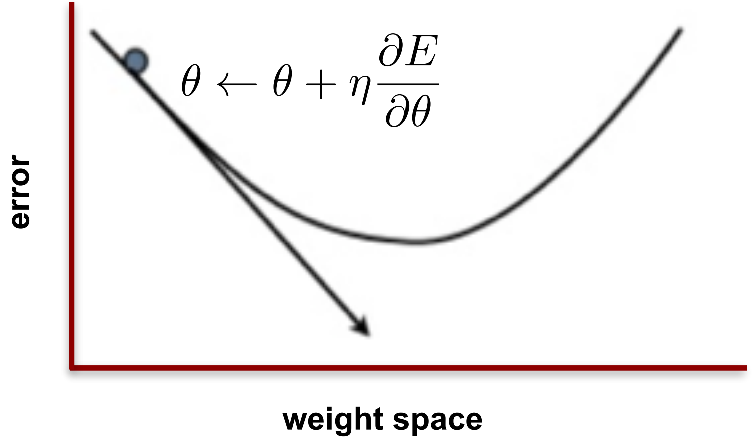
$$\mathbf{h} = \mathbf{f}(\mathbf{W}^1\mathbf{x}) = \begin{bmatrix} f(w^1_{11}x_1 + w^1_{12}x_2 + \cdots + w^1_{1J}x_J) \\ \vdots \\ f(w^1_{I1}x_1 + w^1_{I2}x_2 + \cdots + w^1_{IJ}x_J) \end{bmatrix}$$

- Becomes the input to the next layer
- And so on until we reach the output layer

Learning algorithm: Backpropagation

While not converged:

- Pick random training case (**x**,**y**); e.g. image **x** and its category **y**
- Run neural network on input **x** and compute an output **z**
- Modify connection weights such as to minimize an error function *E*(**y**,**z**)
- Follow gradient of the error w.r.t. the connections (gradient descent)


$\theta \leftarrow \theta + \eta \frac{\partial E}{\partial \theta}$
error
weight space

The universal approximation theorem (informal version)

A feed-forward network with a single hidden layer containing a finite number of neurons (i.e., a multilayer perceptron), can approximate any continuous function (under mild assumptions on the activation function).

Cybenko., G. (1989) "Approximations by superpositions of sigmoidal functions", *Mathematics of Control, Signals, and Systems*, 2 (4), 303-314

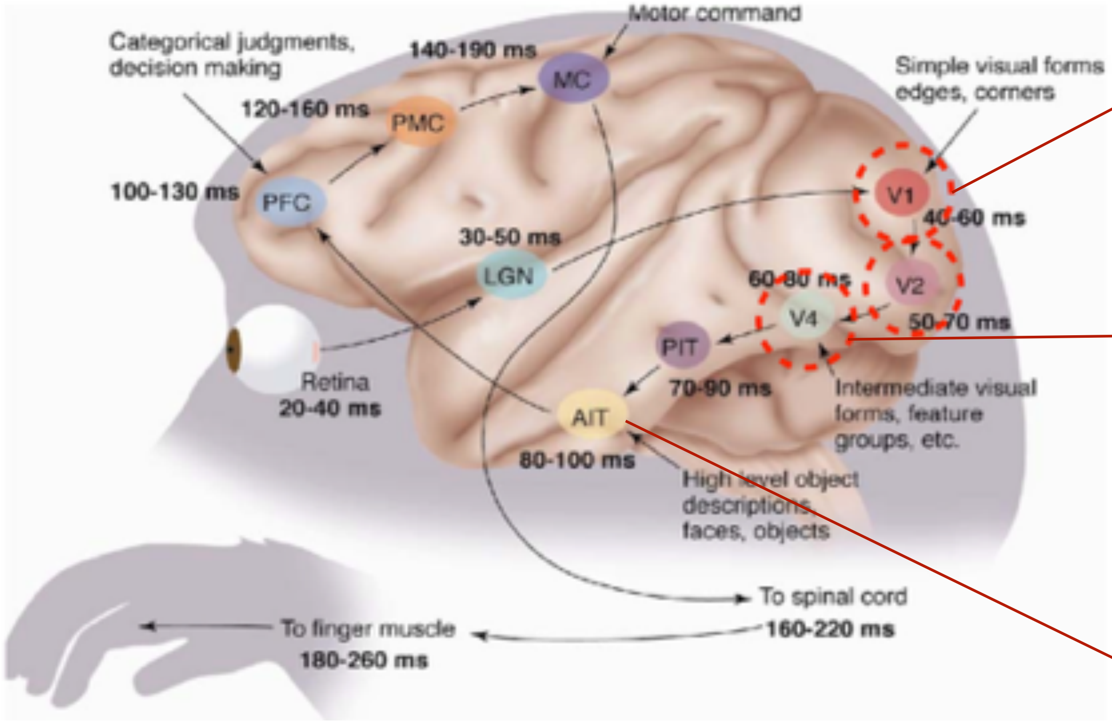
Kurt Hornik (1991) "Approximation capabilities of multilayer feedforward networks", *Neural Networks*, 4(2), 251–257.

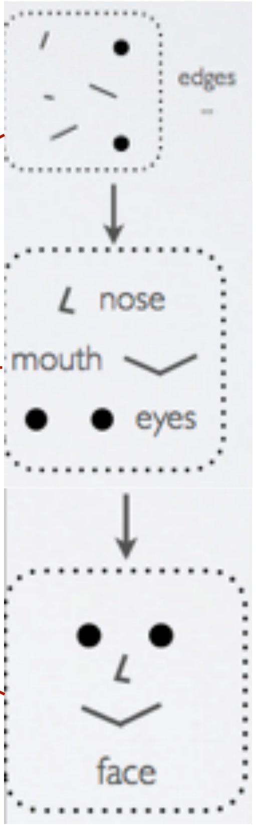
Deep neural networks can represent many functions much more compactly than shallow neural networks

Delalleau & Bengio. Shallow vs. Deep Sum-Product Networks. NIPS

Neuroscientific inspiration

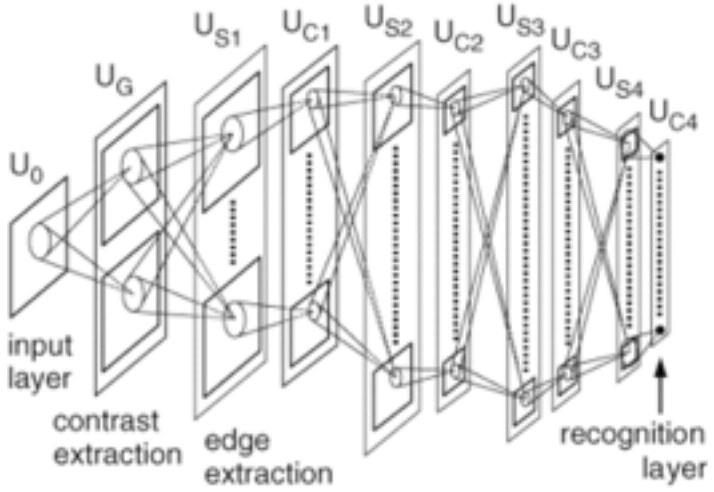
Visual features (Thorpe)



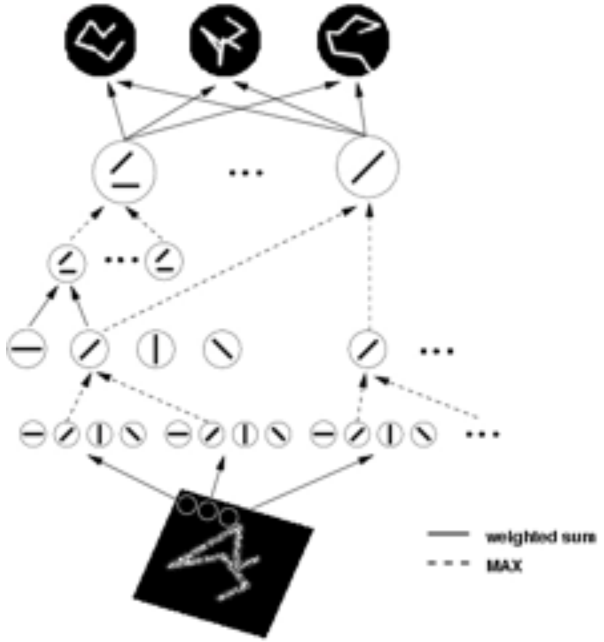


Neuroscientific inspiration

- Classical models of stimulus transformations in sensory cortex

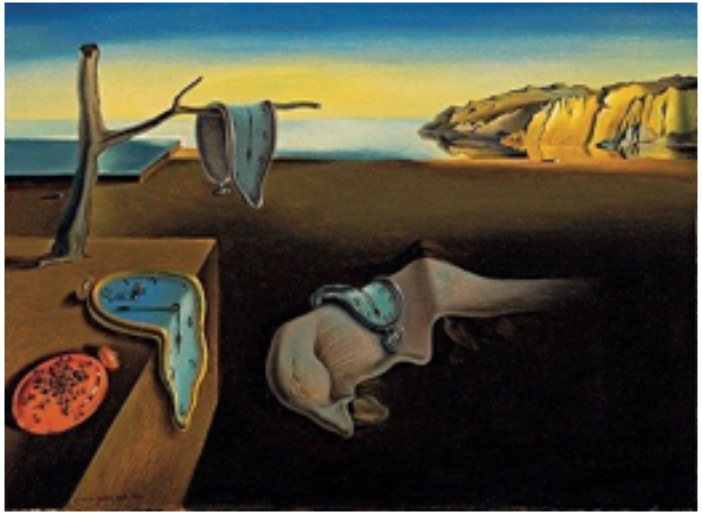


Fukushima, 1980, Biol. Cybern.



Riesenhuber & Poggio, 1999, Nat. Neurosci.

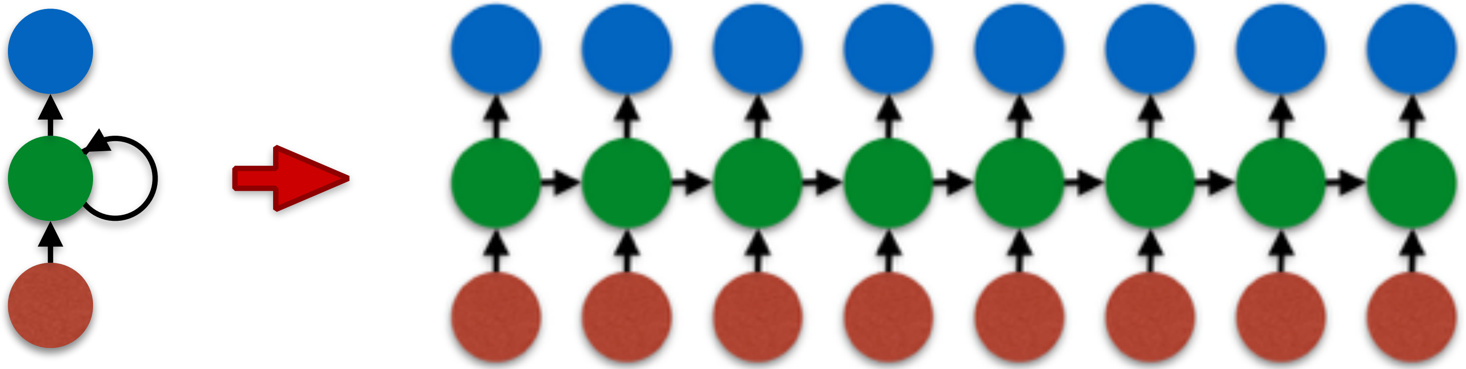
Modeling temporal dependencies


Dali, La persistencia de la memoria

Humans don’t start their thinking from scratch every second. As you read this, you understand each word based on your understanding of previous words. You don’t throw everything away and start thinking from scratch again. Your thoughts have persistence.

Recurrent neural networks

- Feedforward neural networks only capture static functions
- Recurrent neural networks are models of dynamical systems



- Temporal dependencies captured by means of feedback loops
- Inference done by unrolling the network over time and applying backpropagation

Neural Turing machines

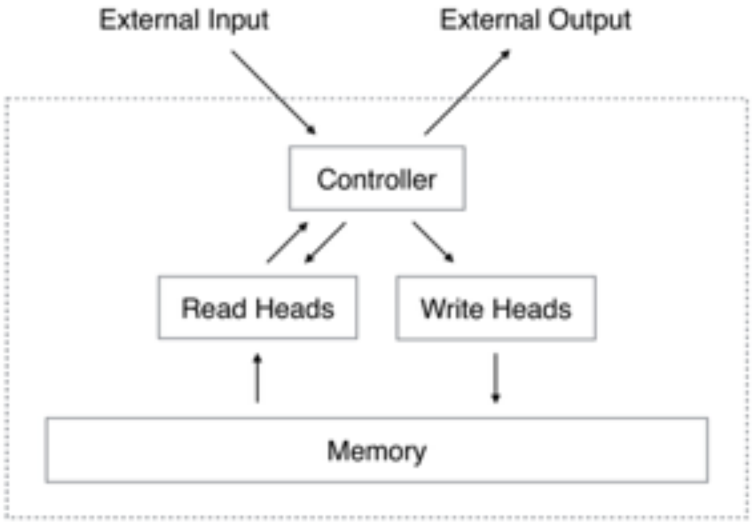
RNNs are Turing-Complete (Siegelmann and Sontag, 1995), and therefore have the capacity to simulate arbitrary procedures, if properly wired

A Neural Turing machine (NTM) is a differentiable computer that can be trained by gradient descent, yielding a practical mechanism for learning programs

An NTM consists of two basic components:

- a neural network controller (typically an LSTM RNN)
- a memory bank

Neural Turing machines



- Fully differentiable model by defining blurry read-write operations
- NTMs allow variable binding and dealing with variable-length structures
- These concepts were previously used as counterarguments for neural networks as models of human cognition (Fodor & Pylyshyn, 1988)

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural Turing Machines, 1–26.

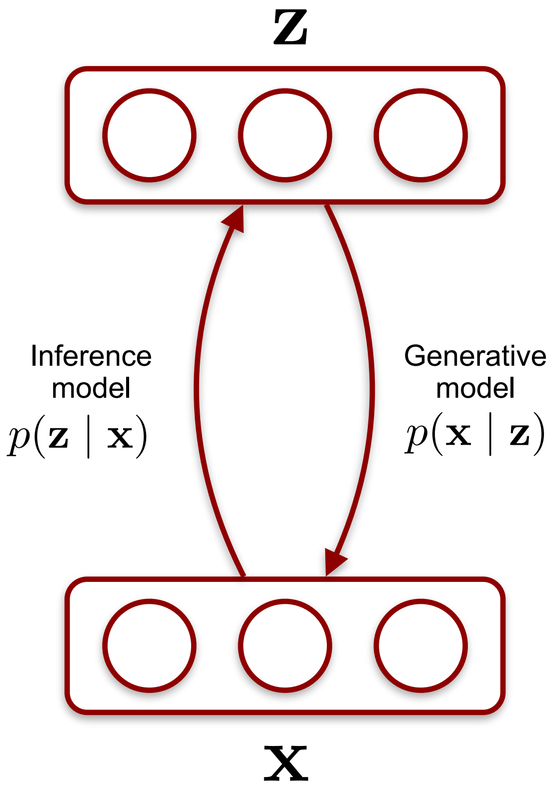
Relevance for neuroscience

- Bayesian model can be seen as normative models that describe how brains should optimally behave
- ANNs can be seen as implementations that show how cognitive tasks can be realized by interacting neuronal populations:
    - deep neural networks that model sensory processing
    - self-organizing maps that implement semantic categorization
    - radial basis function networks that model prototype learning
    - Hopfield networks that model autoassociative memory

Important research topics

- Rate-based versus spiking neural networks
- Biologically plausible implementations (learning rules, objective functions)
- Probabilistic inference by neural networks
- Fundamental work on representation learning
- Generative models that incorporate top-down processing

Example: generative models


$\mathbf{z}$
Inference model
$p(\mathbf{z} \mid \mathbf{x})$
Generative model
$p(\mathbf{x} \mid \mathbf{z})$
$\mathbf{x}$

- Implement bottom-up and top-down drive
- Representation of uncertainty
- Unsupervised rather than supervised learning

Part III: Reinforcement Learning

Reinforcement learning

RL is a general-purpose framework for artificial intelligence

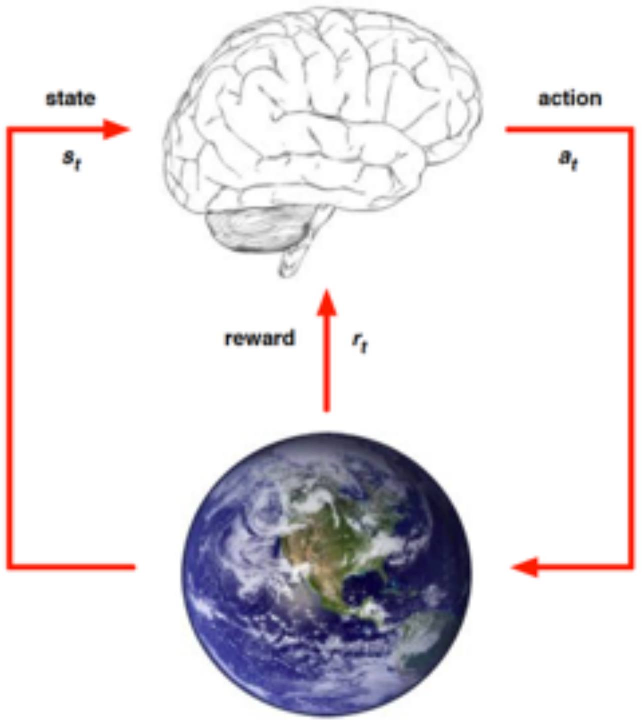
- RL can be used by an agent with the capacity to act
- Each action influences the agent's future state
- Success is measured by a scalar reward signal

RL in a nutshell:

- Select actions to maximise future reward
- The essence of an intelligent agent

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

Deep reinforcement learning



At each step t the agent:

- Receives state s_t
- Receives scalar reward r_t
- Executes action a_t

The environment:

- Receives action a_t
- Emits state s_t
- Emits scalar reward r_t

Policy is a function that selects actions given states: $a = \pi(s)$

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

Approaches to reinforcement learning

Model-free RL:

Policy-based RL

- Search directly for the optimal policy *π**
- This is the policy achieving maximum future reward

Value-based RL

- Estimate the optimal action-value function *Q*(s; a)*
- This is the maximum value achievable under any policy

Model-based RL

- Build a transition model of the environment
- Plan (e.g. by lookahead) using model

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

Example: Q-learning

Action-value function *Q(s; a)* is expected total reward from state *s* and action *a* under policy *π*:

$$Q^\pi(s,a) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s,a]$$

“How good is action a in state s?"

Q learning

The action-value function

$$Q : S \times A \rightarrow R$$

can be learned using a procedure called Q-learning (Watkins, 1989):

- Before learning has started, *Q* returns an (arbitrary) fixed value
- Each time the agent selects an action, and observes a reward and a new state that may depend on both the previous state and the selected action, *Q* is updated.
- The core of the algorithm is a simple value iteration update.
- It assumes the old value and makes a correction based on the new information.

Q learning

Update equation:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \eta \left(\overbrace{r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)}^{\textbf{TD error}} \right)$$

where r_{t+1} is the reward observed after performing a_t in s_t, and where η is the learning rate

Q learning

The action-value function

$$Q : S \times A \rightarrow R$$

is in essence one huge lookup table.

Becomes impossible to maintain for large (real-world) inputs (most states are never seen)

Solution: Represent action-value function by a function parameterized by θ

$$Q(s, a; \theta) \approx Q^{\pi}(s, a)$$

Good parameters may allow us to generalize to states we never encountered.

Deep Q learning

We can choose the parameterized function to be a (deep) neural network

Define objective function by mean-squared TD error:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta) - Q(s_t, a_t; \theta)\right)^2\right]$$

Leads to the following Q-learning gradient

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta} = \mathbb{E}\left[\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta) - Q(s_t, a_t; \theta)\right) \frac{\partial Q(s_t, a_t; \theta)}{\partial \theta}\right]$$

Optimise objective end-to-end by stochastic gradient descent.

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

Stability issues

Naive Q-learning oscillates or diverges with neural nets:

1. Data is sequential

- Successive samples are correlated, non-iid

2. Policy changes rapidly with slight changes to Q-values

- Policy may oscillate
- Distribution of data can swing from one extreme to another

3. Scale of rewards and Q-values is unknown

- Naive Q-learning gradients can be large unstable when backpropagated

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

Deep Q networks

DQN provides a stable solution to deep value-based RL:

1. Use experience replay

- Break correlations in data, bring us back to iid setting
- Learn from all past policies

2. Freeze target Q-network

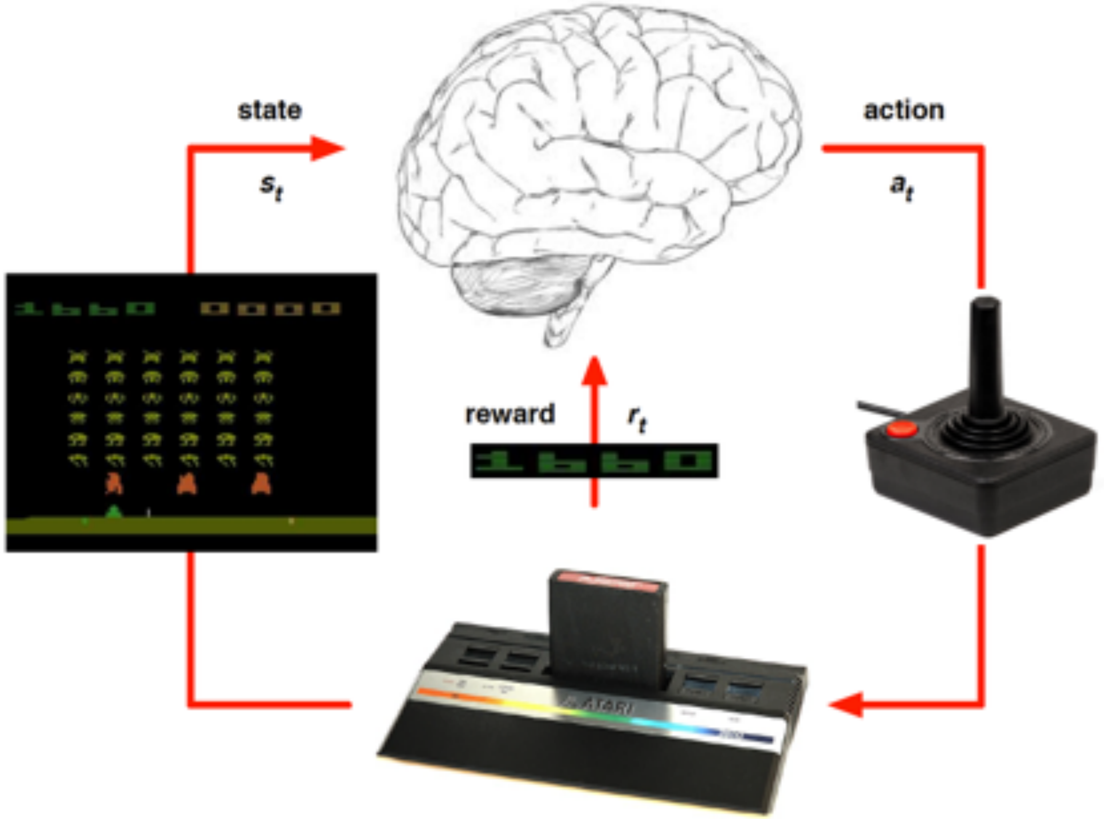
- Avoid oscillations
- Break correlations between Q-network and target

3. Clip rewards or normalize network adaptively to sensible range

- Robust gradients

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

Playing Atari games with deep reinforcement learning

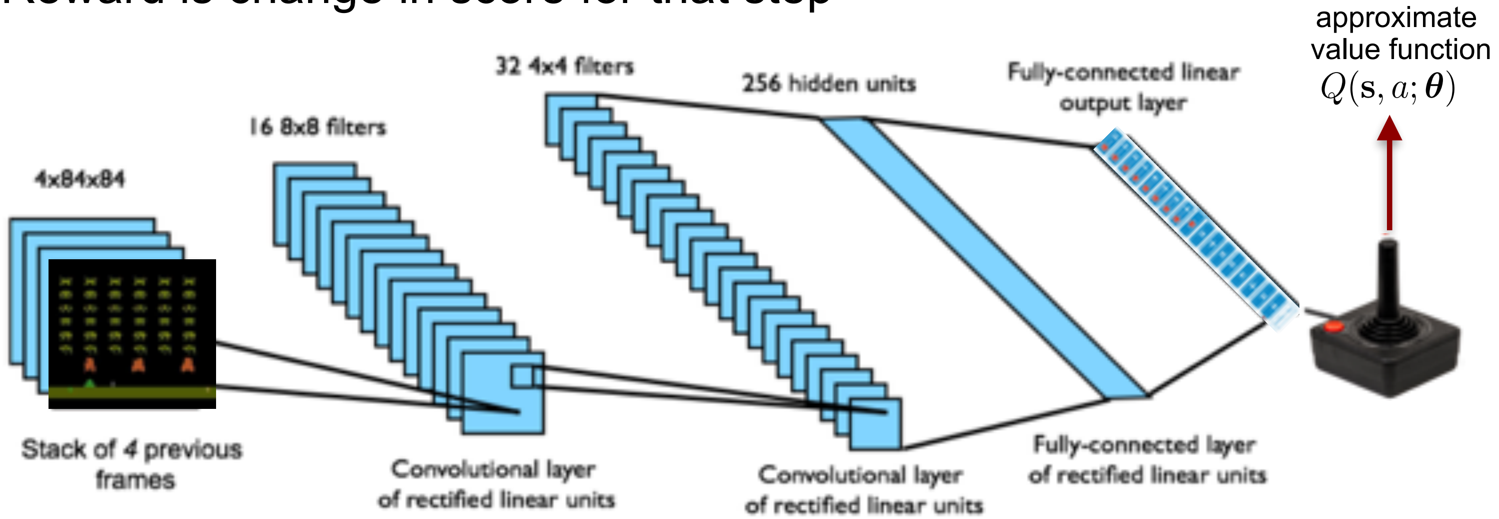


http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

DQN in Atari games

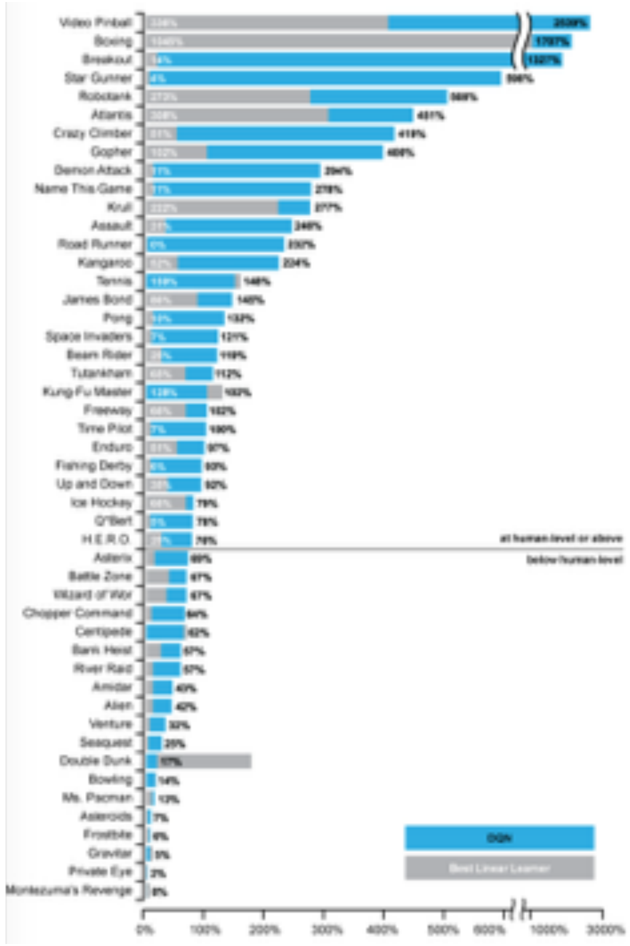
End-to-end learning of values *Q(s; a)* from pixels *s*

- Input state *s* is stack of raw pixels from last 4 frames
- Output is *Q(s; a)* for 18 joystick/button positions
- Reward is change in score for that step

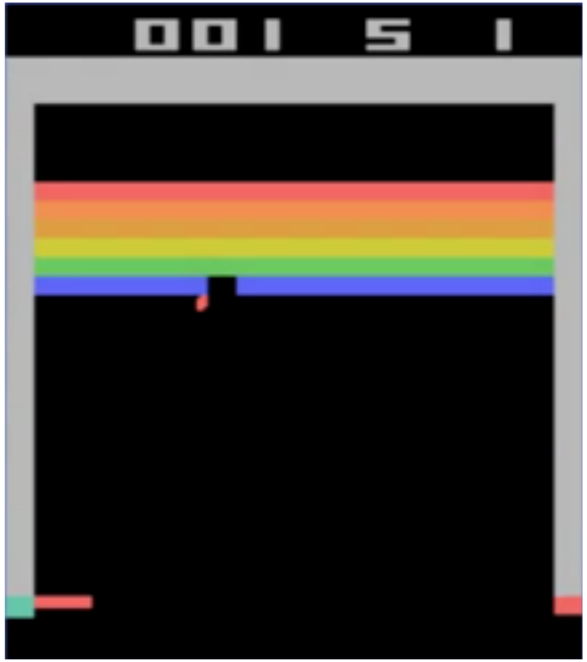

approximate value function
$Q(\mathbf{s}, a; \boldsymbol{\theta})$
32 4x4 filters
256 hidden units
Fully-connected linear output layer
16 8x8 filters
4x84x84
Stack of 4 previous frames
Convolutional layer of rectified linear units
Convolutional layer of rectified linear units
Fully-connected layer of rectified linear units

http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf

DQN results in Atari

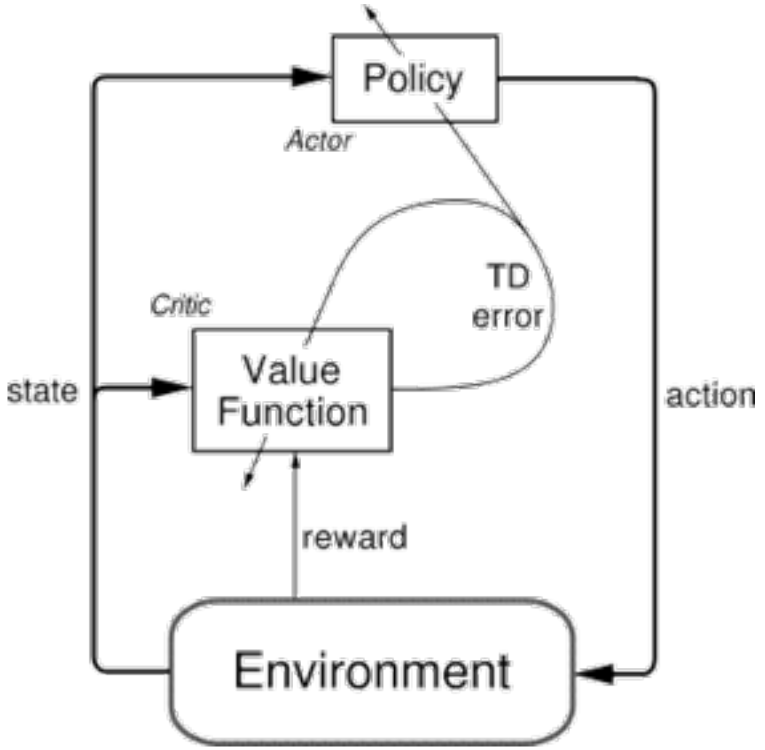


Example: DQN playing Breakout



Mnih et al. (2015). Nature

Example: Actor-critic methods



- On-policy algorithm
- Critic learns the value of being in a particular state

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t),$$

- Actor learns a (continous or discrete) policy

$$\pi_t(s,a) = Pr\{a_t = a \mid s_t = s\} = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}},$$

- Update of the form

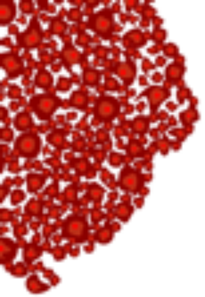
$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t,$$

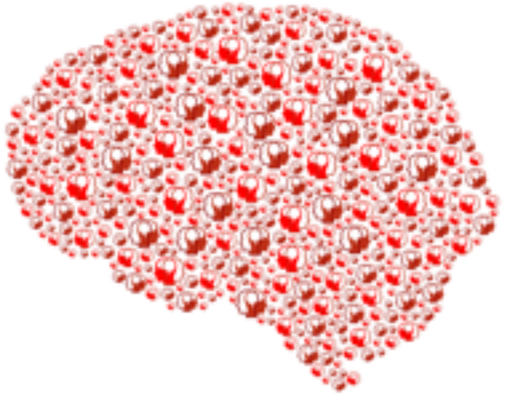
https://webdocs.cs.ualberta.ca/~sutton/book/ebook/node66.html

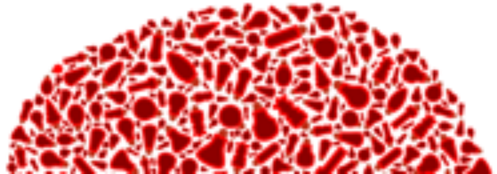
Practical work

- Three practical assignments
  - Implement beta-binomial model
  - Implement multilayer perceptron in chainer or tensorflow
  - Implement Tabular Q-learning for a simple environment

- Implementation in Python
- Pass for all assignments prerequisite for exam participation







www.ru.nl/donders

www.ccnlab.net