Please pick up an anonymous evaluation sheet

Homework 3 Questions

1 K-means

Run examples 1 through 3 in the attached code. Each example displays a different result of running the k-means clustering algorithm on the iris data set. Which result do you think is the best, and why do the other runs of k-means fail?

2 PCA

How many principal components (PCs) could possibly be in the iris data set?

Run example 4 in the attached code. How much of the variance in the data set is explained by the first PC?

Run example 5 in the attached code. Does including another component improve the results? Why or why not?

Question 1: When doing PCA, using a higher number of PCs to create the final reduced representation of the data would __________ overfitting.

a. increase

b. decrease

Question 2: CCA can be used to find the linear transformations of more than 2 data sets that are most correlated.

a. true

b. false

Question 3: You want to perform k-means 2 separate times. You have asked for 2 clusters. The data also has labels, though note that k-means is still unsupervised (i.e. does not make use of the labels). After completing the first k-means with k-means++ initialization, you observe that all data from class 1 is in cluster 1. Now, you run k-means a second time, but you specify the means of the clusters to be the means of the clusters obtained after the first run of k-means. Do you expect that all data points from class 1 will again be in cluster 1?

a. yes

b. no

Question 4: Now, instead of running a second k-means with the cluster means initialized to be the cluster means from the first run, you run a second k-means with k-means++ initialization. Do you expect that all data points from class 1 will again be in cluster 1?

a. yes

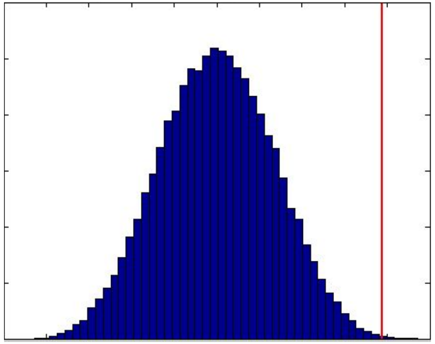
b. no

Advanced topics:
latent variable models, deep learning, and reinforcement learning

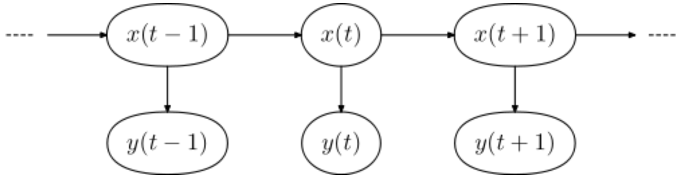
06/24/2016
Mariya Toneva
mariya@cmu.edu

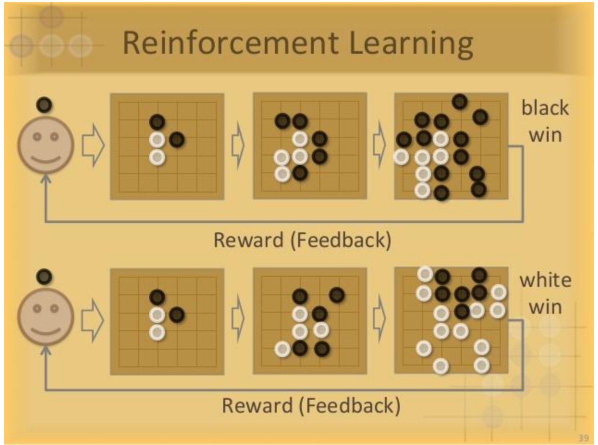
Some figures derived from slides by Tom Mitchell, Aarti Singh, Barnabás Póczos

How can ML help neuroscientists?



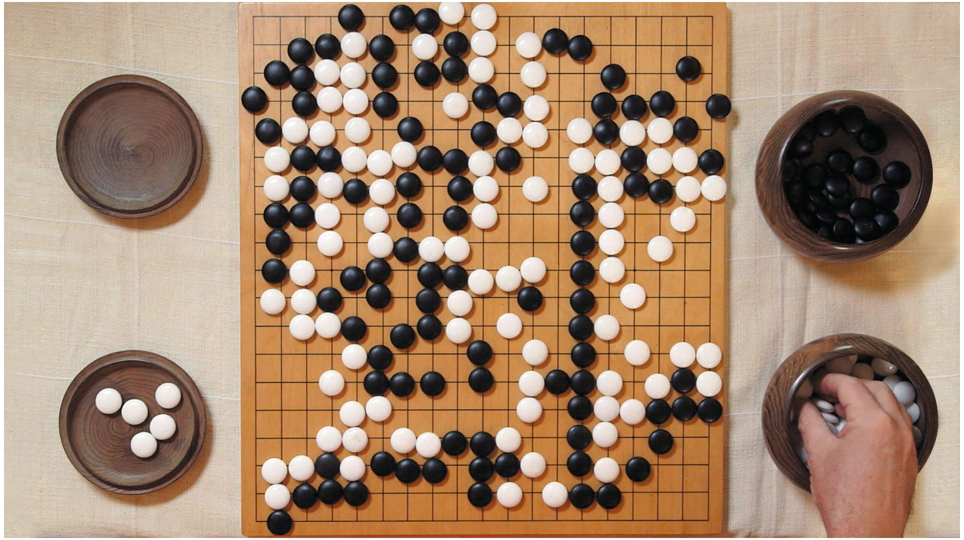
- ❏ Evaluate results
  - ❏ cross validation (how generalizable are our results?)
  - ❏ nearly assumption-free significance testing (are the results different from chance?)
- ❏ **Complex data-driven hypotheses of brain processing**
  - ❏ **advanced topics: latent variable models, reinforcement learning, deep learning**





Look ahead to end of lecture: understanding the most recent AI feat of beating Go world champion

Mnih et al., 2013



Silver et al., 2016 Nature

Today: advanced topics!

- Latent variable models
  - Hidden Markov models
- Reinforcement learning
- Deep learning
  - Neural networks
    - Recurrent NNs
      - Long short-term memory networks
  - Deep belief networks
  - Convolutional neural networks
- AlphaGo: RL + deep learning!

Importance of sequential data

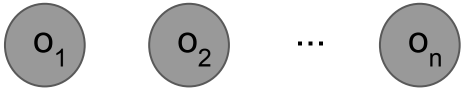
- Moving from treating all observations as independent from each other, to dependent

Importance of sequential data

- Moving from treating all observations as independent from each other, to dependent
- Important, when considering sequential data, like we have in neuroscience:

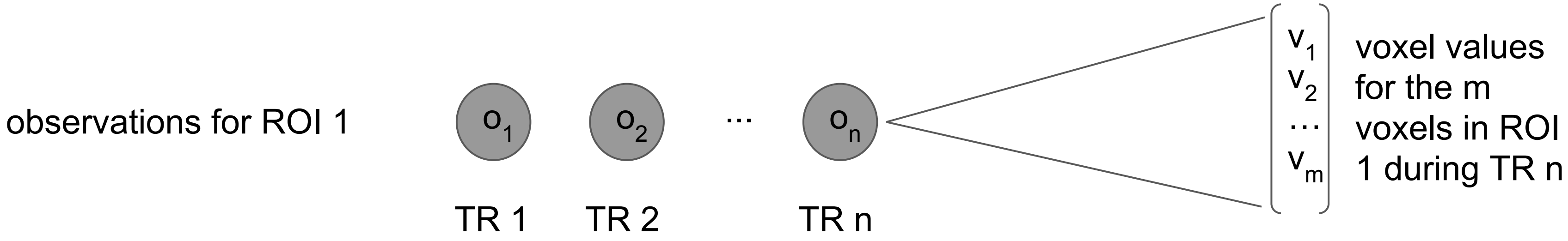
Importance of sequential data

- Moving from treating all observations as independent from each other, to dependent
- Important, when considering sequential data, like we have in neuroscience:
  - Example: observation 1 = fMRI voxel activities in ROI 1 for TR 1
    observation 2 = fMRI voxel activities in ROI 1 for TR 2
    …
    observation N = fMRI voxel activities in ROI 1 for TR N



Importance of sequential data

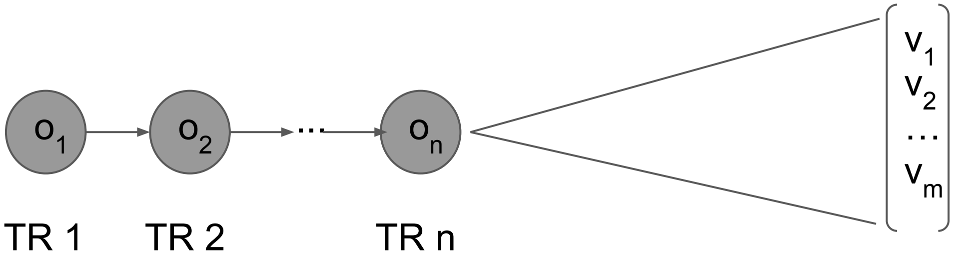
- Moving from treating all observations as independent from each other, to dependent
- Important, when considering sequential data, like we have in neuroscience:
  - Example: observation 1 = fMRI voxel activities in ROI 1 for TR 1
    observation 2 = fMRI voxel activities in ROI 1 for TR 2
    …
    observation N = fMRI voxel activities in ROI 1 for TR N



Importance of sequential data

- Moving from treating all observations as independent from each other, to dependent
- Important, when considering sequential data, like we have in neuroscience:
  - Example: observation 1 = fMRI voxel activities in ROI 1 for TR 1
    observation 2 = fMRI voxel activities in ROI 1 for TR 2
    …
    observation N = fMRI voxel activities in ROI 1 for TR N

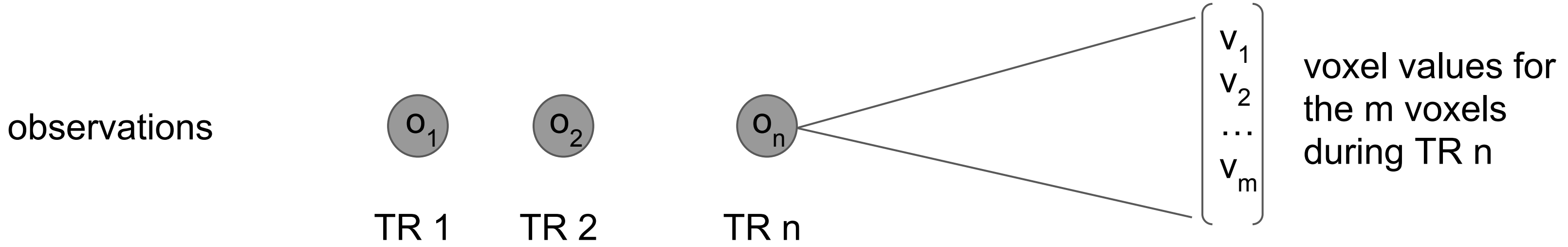
observations for ROI 1



voxel values for the m voxels in ROI 1 during TR n

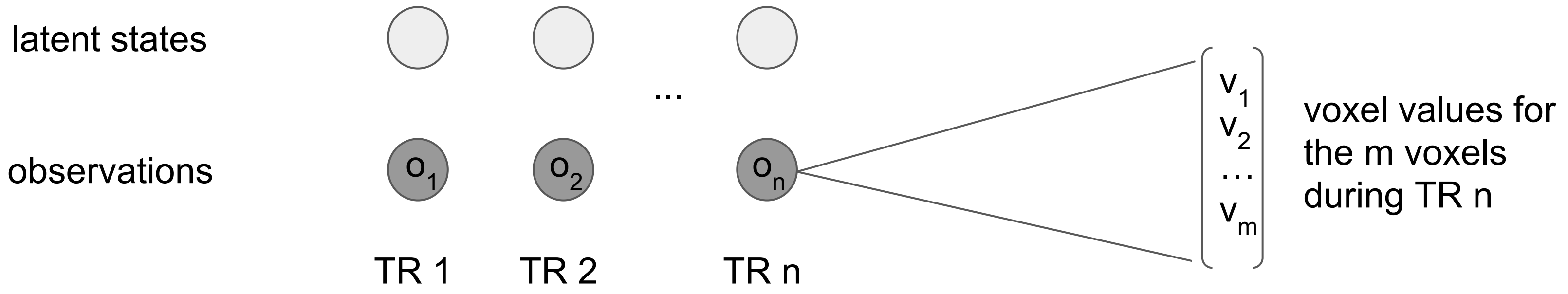
What if there are latent processes?

- ❑ Dependence in observations may be due to dependence in latent states



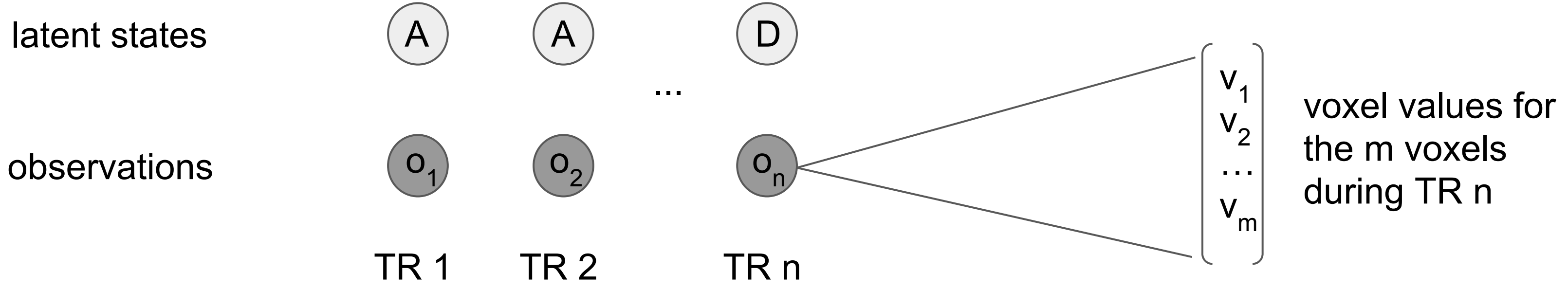
What if there are latent processes?

- ❑ Dependence in observations may be due to dependence in latent states
  - ❑ Example:
    - ❑ underlying processes while subject is at rest in an fMRI



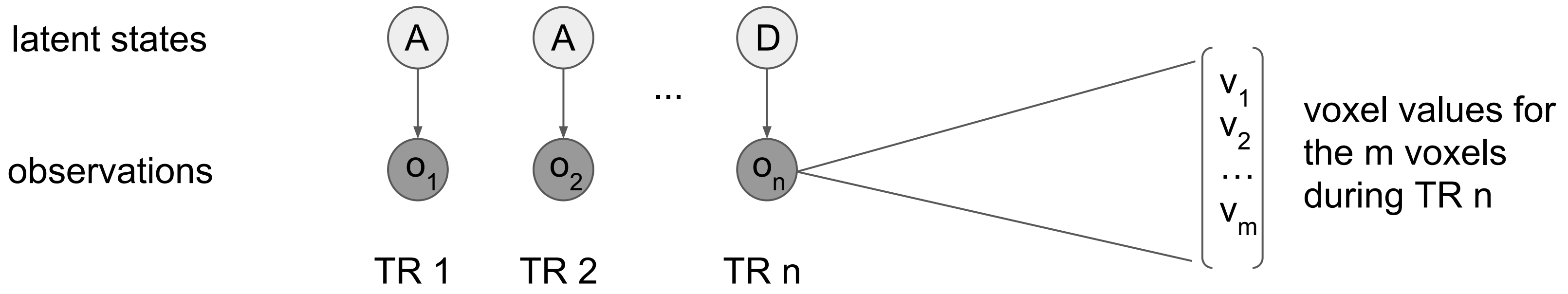
What if there are latent processes?

- Dependence in observations may be due to dependence in latent states
  - Example:
    - underlying processes while subject is at rest in an fMRI
      - “attentive state” (A)
      - “default state” (D)



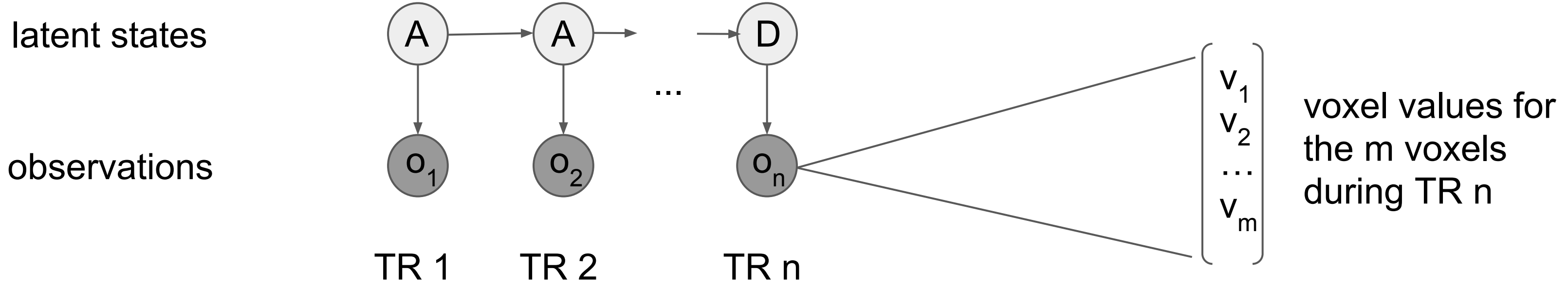
What if there are latent processes?

- Dependence in observations may be due to dependence in latent states
  - Example:
    - underlying processes while subject is at rest in an fMRI
      - “attentive state” (A)
      - “default state” (D)
    - voxel values for ROIs in certain TRs may depend on whether subject is in state A or D

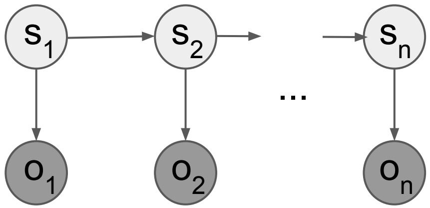


What if there are latent processes?

- Dependence in observations may be due to dependence in latent states
    - Example:
        - underlying processes while subject is at rest in an fMRI
            - "attentive state" (A)
            - "default state" (D)
        - voxel values for ROIs in certain TRs may depend on whether subject is in state A or D
        - transitioning between states may help predict voxel values during next TR

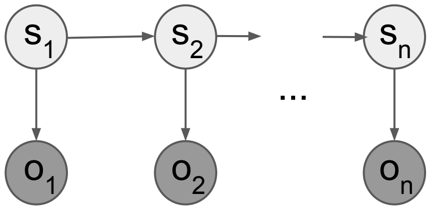


Hidden Markov model (HMM)



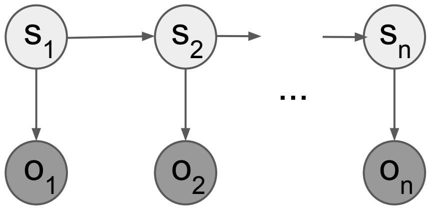
- Observation space $O_t \in \{x_1, x_2, ..., x_k\}$
  - observed data
  - can be discrete or real-valued
  - example: voxel values for ROI 1 during one TR

Hidden Markov model (HMM)



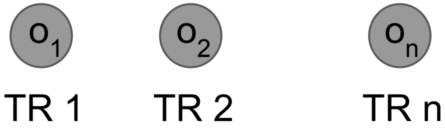
- Observation space $O_t \in \{x_1, x_2, ..., x_k\}$
  - observed data
  - can be discrete or real-valued
  - example: voxel values for ROI 1 during one TR
- Hidden state space $S_t \in \{1, ..., m\}$
  - latent states
  - discrete
  - example: A (attentive state), D (default state)

Hidden Markov model (HMM)



- Observation space $O_t \in \{x_1, x_2, ..., x_k\}$
  - observed data
  - can be discrete or real-valued
  - example: voxel values for ROI 1 during one TR
- Hidden state space $S_t \in \{1, ..., m\}$
  - latent states
  - discrete
  - example: A (attentive state), D (default state)
- Observed states are usually shaded

What can we use HMMs for?

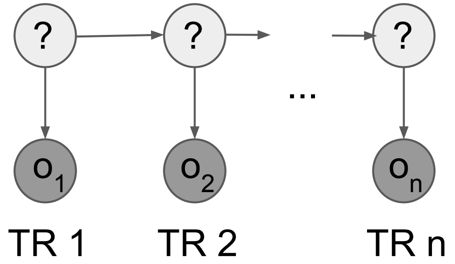


- We start with only having observations
  - e.g. voxel values for ROI 1 during all TRs

What can we use HMMs for?

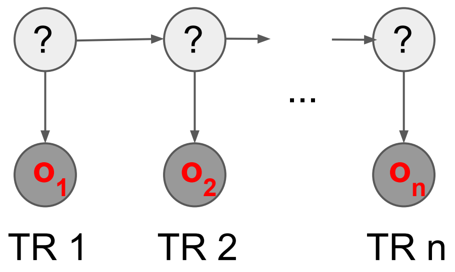
- We start with only having observations
  - e.g. voxel values for ROI 1 during all TRs
- Test a hypothesis about the underlying processes in the brain that lead to the observations

What can we use HMMs for?



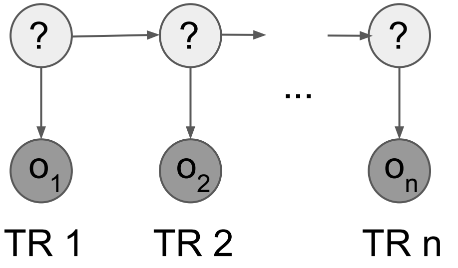
- We start with only having observations
  - e.g. voxel values for ROI 1 during all TRs
- Test a hypothesis about the underlying processes in the brain that lead to the observations
  - hypothesis: at rest, the brain switches between an attentive and default state

What can we use HMMs for?



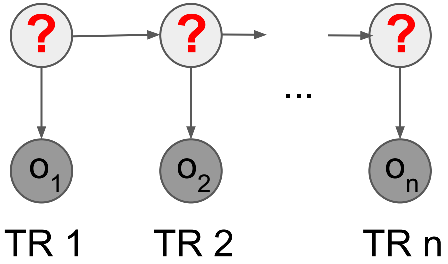
- We start with only having observations
  - e.g. voxel values for ROI 1 during all TRs
- Test a hypothesis about the underlying processes in the brain that lead to the observations
  - hypothesis: at rest, the brain switches between an attentive and default state
  - test with HMM: given our hypothesis, how likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

What can we use HMMs for?



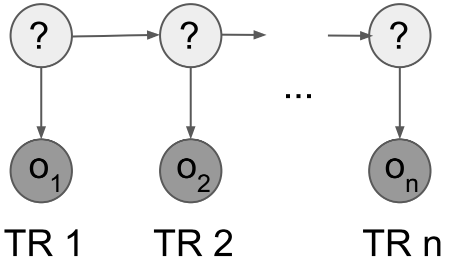
- We start with only having observations
  - e.g. voxel values for ROI 1 during all TRs
- Test a hypothesis about the underlying processes in the brain that lead to the observations
  - hypothesis: at rest, the brain switches between an attentive and default state
  - test with HMM: given our hypothesis, how likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?
- Infer the best sequence of hidden states that led to the observations

What can we use HMMs for?



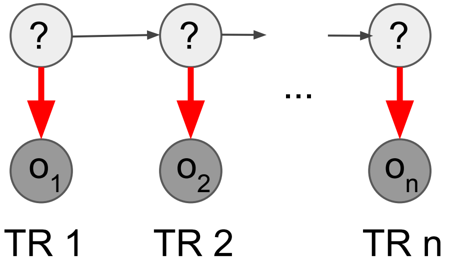
- We start with only having observations
  - e.g. voxel values for ROI 1 during all TRs
- Test a hypothesis about the underlying processes in the brain that lead to the observations
  - hypothesis: at rest, the brain switches between an attentive and default state
  - test with HMM: given our hypothesis, how likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?
- Infer the best sequence of hidden states that led to the observations
  - Inferring what brain state the subject was in during each TR

What can we use HMMs for?



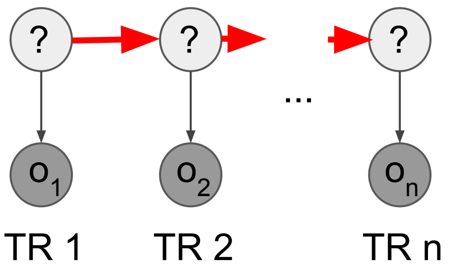
- We start with only having observations
  - e.g. voxel values for ROI 1 during all TRs
- Test a hypothesis about the underlying processes in the brain that lead to the observations
  - hypothesis: at rest, the brain switches between an attentive and default state
  - test with HMM: given our hypothesis, how likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?
- Infer the best sequence of hidden states that led to the observations
  - Inferring what brain state the subject was in during each TR
- How are observations emitted from the hidden states? How does the model transition from one hidden state to another?

What can we use HMMs for?



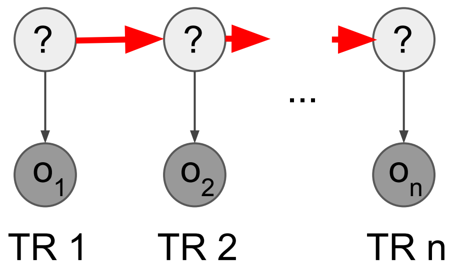
- We start with only having observations
  - e.g. voxel values for ROI 1 during all TRs
- Test a hypothesis about the underlying processes in the brain that lead to the observations
  - hypothesis: at rest, the brain switches between an attentive and default state
  - test with HMM: given our hypothesis, how likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?
- Infer the best sequence of hidden states that led to the observations
  - Inferring what brain state the subject was in during each TR
- How are observations emitted from the hidden states? How does the model transition from one hidden state to another?
  - Learn how the observed voxel values for a TR are related to the current brain state

What can we use HMMs for?



- We start with only having observations
  - e.g. voxel values for ROI 1 during all TRs
- Test a hypothesis about the underlying processes in the brain that lead to the observations
  - hypothesis: at rest, the brain switches between an attentive and default state
  - test with HMM: given our hypothesis, how likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?
- Infer the best sequence of hidden states that led to the observations
  - Inferring what brain state the subject was in during each TR
- How are observations emitted from the hidden states? How does the model transition from one hidden state to another?
  - Learn how the observed voxel values for a TR are related to the current brain state
  - Learn how the brain transitions between attentive and default states

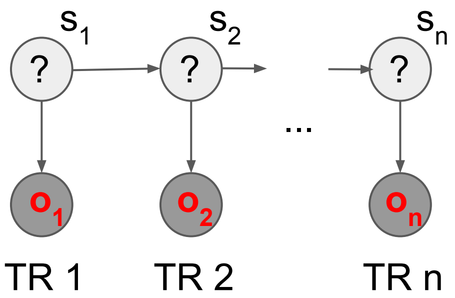
What can we use HMMs for?



- We start with only having observations
  - e.g. voxel values for ROI 1 during all TRs
- Test a hypothesis about the underlying processes in the brain that lead to the observations **Forward algorithm**
  - hypothesis: at rest, the brain switches between an attentive and default state
  - test with HMM: given our hypothesis, how likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?
- Infer the best sequence of hidden states that led to the observations
  - Inferring what brain state the subject was in during each TR **Viterbi algorithm**
- How are observations emitted from the hidden states? How does the model transition from one hidden state to another? **Expectation-Maximization (EM)**
  - Learn how the observed voxel values for a TR are related to the current brain state
  - Learn how the brain transitions between attentive and default states

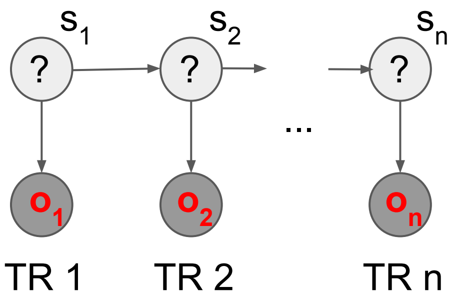
How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)



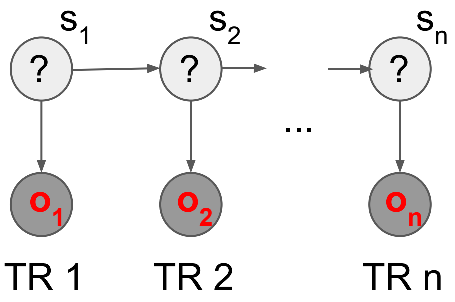
How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)
- Assume that we already learned both the emissions $P(o_i|s_i)$ and transitions $P(s_{i+1}|s_i)$ with EM



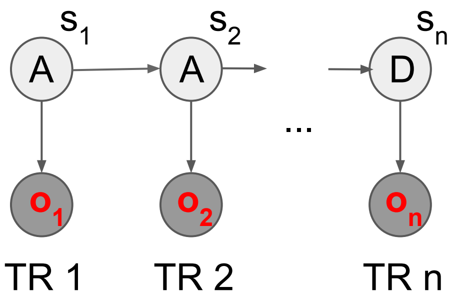
How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)
- Assume that we already learned both the emissions $P(o_i|s_i)$ and transitions $P(s_{i+1}|s_i)$ with EM
- Want to evaluate $P(o_1,...,o_n)$ under this model



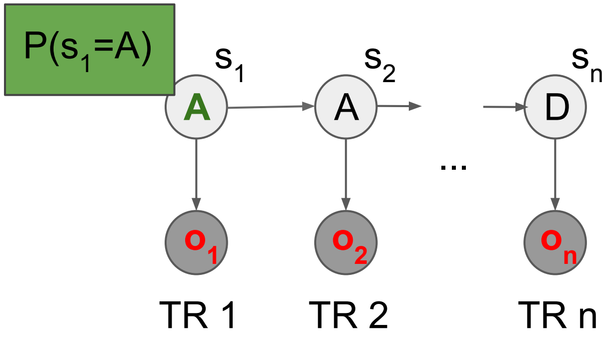
How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- ❑ Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)
- ❑ Assume that we already learned both the emissions $P(o_i|s_i)$ and transitions $P(s_{i+1}|s_i)$ with EM
- ❑ Want to evaluate $P(o_1,...,o_n)$ under this model
- ❑ If we know what the values of $s_1,...,s_n$ are, we quickly evaluate $P(o_1,...,o_n)$
  - ❑ example: if s_1= A, s_2= A,...,s_3= D



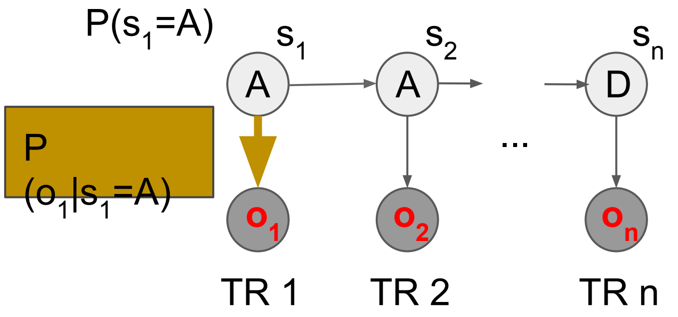
How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)
- Assume that we already learned both the emissions $P(o_i|s_i)$ and transitions $P(s_{i+1}|s_i)$ with EM
- Want to evaluate $P(o_1,...,o_n)$ under this model
- If we know what the values of $s_1,...,s_n$ are, we quickly evaluate $P(o_1,...,o_n)$
  - example: if s_1= A, s_2= A,...,s_3= D



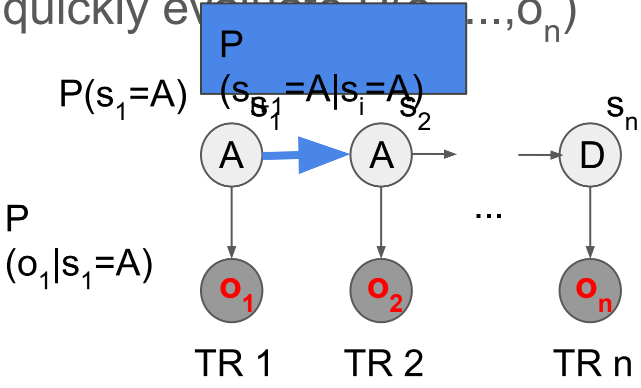
How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)
- Assume that we already learned both the emissions $P(o_i|s_i)$ and transitions $P(s_{i+1}|s_i)$ with EM
- Want to evaluate $P(o_1,...,o_n)$ under this model
- If we know what the values of $s_1,...,s_n$ are, we quickly evaluate $P(o_1,...,o_n)$
  - example: if s_1= A, s_2= A,...,s_3= D



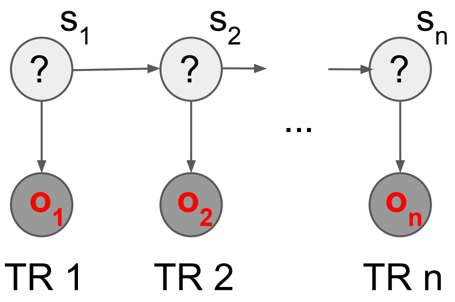
How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)
- Assume that we already learned both the emissions $P(o_i|s_i)$ and transitions $P(s_{i+1}|s_i)$ with EM
- Want to evaluate $P(o_1,...,o_n)$ under this model
- If we know what the values of $s_1,...,s_n$ are, we quickly evaluate $P(o_1,...,o_n)$
  - example: if s_1= A, s_2= A,...,s_3= D



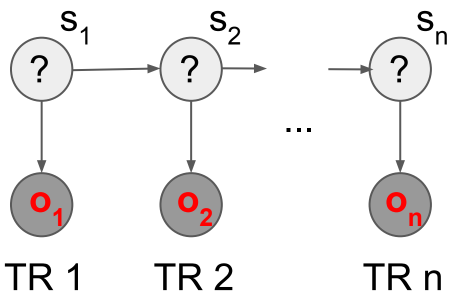
How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)
- Assume that we already learned both the emissions $P(o_i|s_i)$ and transitions $P(s_{i+1}|s_i)$ with EM
- Want to evaluate $P(o_1,...,o_n)$ under this model
- If we know what the values of $s_1,...,s_n$ are, we quickly evaluate $P(o_1,...,o_n)$
  - example: if s_1= A, s_2= A,...,s_3= D
- But we don't know what the hidden state values are!



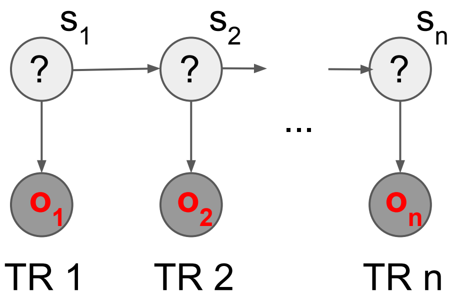
How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)
- Assume that we already learned both the emissions $P(o_i|s_i)$ and transitions $P(s_{i+1}|s_i)$ with EM
- Want to evaluate $P(o_1,...,o_n)$ under this model
- If we know what the values of $s_1,...,s_n$ are, we quickly evaluate $P(o_1,...,o_n)$
  - example: if s_1= A, s_2= A,...,s_3= D
- But we don't know what the hidden state values are!
- So how can we evaluate $P(o_1,...,o_n)$?



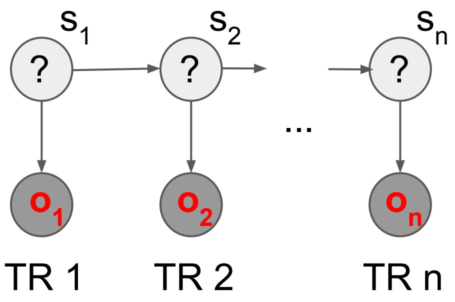
How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)
- Assume that we already learned both the emissions $P(o_i|s_i)$ and transitions $P(s_{i+1}|s_i)$ with EM
- Want to evaluate $P(o_1,...,o_n)$ under this model
- If we know what the values of $s_1,...,s_n$ are, we quickly evaluate $P(o_1,...,o_n)$
  - example: if s_1= A, s_2= A,...,s_3= D
- But we don't know what the hidden state values are!
- So how can we evaluate $P(o_1,...,o_n)$?
  - calculate $P(o_1,...,o_n)$ under all possibilities for $\{s_1,...,s_n\}$ and sum them since we don't know which $\{s_1,...,s_n\}$ is correct



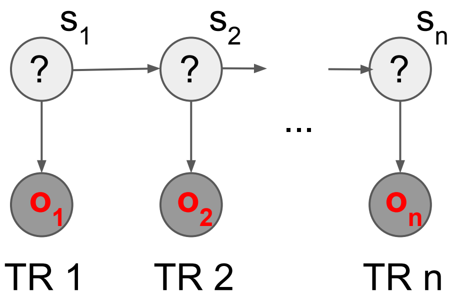
How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)
- Assume that we already learned both the emissions $P(o_i|s_i)$ and transitions $P(s_{i+1}|s_i)$ with EM
- Want to evaluate $P(o_1,...,o_n)$ under this model
- If we know what the values of $s_1,...,s_n$ are, we quickly evaluate $P(o_1,...,o_n)$
  - example: if s_1= A, s_2= A,...,s_3= D
- But we don't know what the hidden state values are!
- So how can we evaluate $P(o_1,...,o_n)$?
  - calculate $P(o_1,...,o_n)$ under all possibilities for $\{s_1,...,s_n\}$ and sum them since we don't know which $\{s_1,...,s_n\}$ is correct
- How many possibilities for $\{s_1,...,s_n\}$?



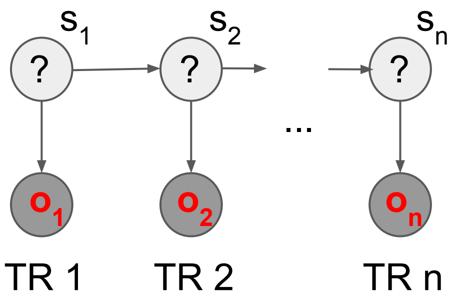
How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)
- Assume that we already learned both the emissions $P(o_i|s_i)$ and transitions $P(s_{i+1}|s_i)$ with EM
- Want to evaluate $P(o_1,...,o_n)$ under this model
- If we know what the values of $s_1,...,s_n$ are, we quickly evaluate $P(o_1,...,o_n)$
  - example: if s_1= A, s_2= A,...,s_3= D
- But we don't know what the hidden state values are!
- So how can we evaluate $P(o_1,...,o_n)$?
  - calculate $P(o_1,...,o_n)$ under all possibilities for $\{s_1,...,s_n\}$ and sum them since we don't know which $\{s_1,...,s_n\}$ is correct
- How many possibilities for $\{s_1,...,s_n\}$? 2^n



How likely is the sequence of observed ROI voxel values $\{o_1,..,o_n\}$?

- Hidden states $\{s_1,...,s_n\}$ can take 2 values: A (attentive) or D (default)
- Assume that we already learned both the emissions $P(o_i|s_i)$ and transitions $P(s_{i+1}|s_i)$ with EM
- Want to evaluate $P(o_1,...,o_n)$ under this model
- If we know what the values of $s_1,...,s_n$ are, we quickly evaluate $P(o_1,...,o_n)$
  - example: if s_1= A, s_2= A,...,s_3= D
- But we don't know what the hidden state values are!
- So how can we evaluate $P(o_1,...,o_n)$?
  - calculate $P(o_1,...,o_n)$ under all possibilities for $\{s_1,...,s_n\}$ and sum them since we don't know which $\{s_1,...,s_n\}$ is correct
- How many possibilities for $\{s_1,...,s_n\}$? 2^n
  - forward algorithm is a clever way to do this

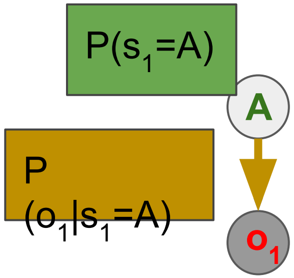


Forward algorithm: computationally friendly way to calculate probability of observed sequence

- Key idea: recognize that there is a lot of overlap between possibilities for $\{s_1,...,s_n\}$, so instead of recalculating the whole probability for each possibility, store probabilities for smaller pieces of the sequence and reuse

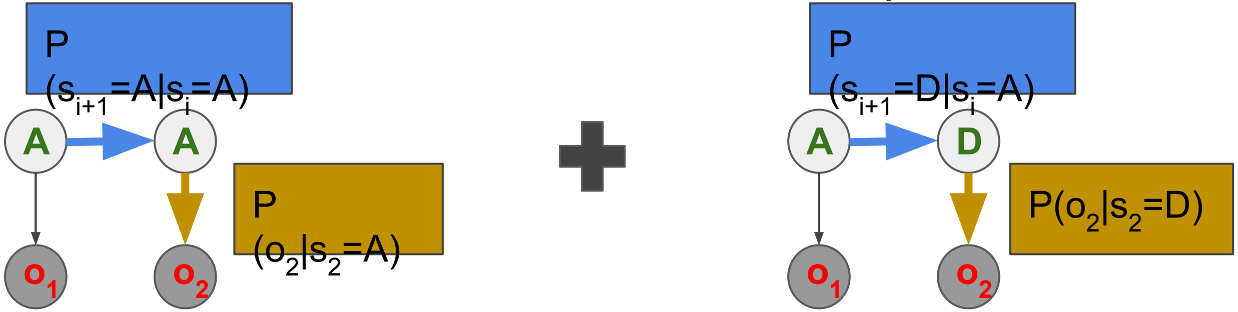
Forward algorithm: computationally friendly way to calculate probability of observed sequence

- Key idea: recognize that there is a lot of overlap between possibilities for $\{s_1,...,s_n\}$, so instead of recalculating the whole probability for each possibility, store probabilities for smaller pieces of the sequence and reuse
- First subunits: **$P(s_1=A)P(o_1|s_1=A)$** and $P(s_1=D)P(o_1|s_1=D)$



Forward algorithm: computationally friendly way to calculate probability of observed sequence

- Key idea: recognize that there is a lot of overlap between possibilities for $\{s_1,\ldots,s_n\}$, so instead of recalculating the whole probability for each possibility, store probabilities for smaller pieces of the sequence and reuse
- First subunits: $P(s_1=A)P(o_1|s_1=A)$ and $P(s_1=D)P(o_1|s_1=D)$
- All further subunits: sum of **emission** x **transition** x previous subunit



Problem 2: infer best sequence of hidden states that results in the sequence of observations

- Viterbi algorithm: also relies on recursive computation using subunits

Problem 2: infer best sequence of hidden states that results in the sequence of observations

- Viterbi algorithm: also relies on recursive computation using subunits
- Starting with the second hidden state, for each hidden state until the last one, we calculate the probability of the most likely path that ends in this hidden state

Problem 2: infer best sequence of hidden states that results in the sequence of observations

- Viterbi algorithm: also relies on recursive computation using subunits
- Starting with the second hidden state, for each hidden state until the last one, we calculate the probability of the most likely path that ends in this hidden state
- For each of these probabilities, we record the previous hidden state in the most likely sequence => this enables us to follow the most probable path backwards once we're at the end of the sequence

Problem 3: learning HMM emissions and transitions

- So far, we've assumed that the emission and transition probabilities were already learned

Problem 3: learning HMM emissions and transitions

- So far, we’ve assumed that the emission and transition probabilities were already learned
- But how do we do this? Expectation-maximization (EM)!

Problem 3: learning HMM emissions and transitions

- So far, we’ve assumed that the emission and transition probabilities were already learned
- But how do we do this? Expectation-maximization (EM)!
- Start with random initializations of parameters, then iterate 2 steps:

Problem 3: learning HMM emissions and transitions

- So far, we’ve assumed that the emission and transition probabilities were already learned
- But how do we do this? Expectation-maximization (EM)!
- Start with random initializations of parameters, then iterate 2 steps:
  - E-step: fix parameters, find expected state assignments

Problem 3: learning HMM emissions and transitions

- So far, we've assumed that the emission and transition probabilities were already learned
- But how do we do this? Expectation-maximization (EM)!
- Start with random initializations of parameters, then iterate 2 steps:
  - E-step: fix parameters, find expected state assignments
  - M-step: fix expected state assignments, update parameters

Problem 3: learning HMM emissions and transitions

- So far, we've assumed that the emission and transition probabilities were already learned
- But how do we do this? Expectation-maximization (EM)!
- Start with random initializations of parameters, then iterate 2 steps:
  - E-step: fix parameters, find expected state assignments
  - M-step: fix expected state assignments, update parameters
- Very general algorithm for learning parameters in models with latent variables

Latent variable models takeaways

- Used to analyze sequential data (such as time series)

Latent variable models takeaways

- Used to analyze sequential data (such as time series)
- Hidden Markov model is an example of such a model, where the hidden states can take discrete values

Latent variable models takeaways

- Used to analyze sequential data (such as time series)
- Hidden Markov model is an example of such a model, where the hidden states can take discrete values
- We can use HMMs for 3 main purposes:

Latent variable models takeaways

- ❏ Used to analyze sequential data (such as time series)
- ❏ Hidden Markov model is an example of such a model, where the hidden states can take discrete values
- ❏ We can use HMMs for 3 main purposes:
  - ❏ Testing a hypothesis about underlying processes in the brain

Latent variable models takeaways

- Used to analyze sequential data (such as time series)
- Hidden Markov model is an example of such a model, where the hidden states can take discrete values
- We can use HMMs for 3 main purposes:
  - Testing a hypothesis about underlying processes in the brain
  - Inferring the best sequence of underlying processes in the brain that emits the observations

Latent variable models takeaways

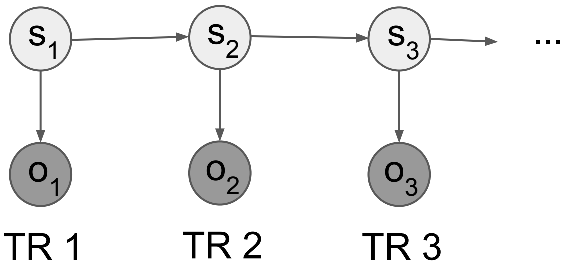
- Used to analyze sequential data (such as time series)
- Hidden Markov model is an example of such a model, where the hidden states can take discrete values
- We can use HMMs for 3 main purposes:
  - Testing a hypothesis about underlying processes in the brain
  - Inferring the best sequence of underlying processes in the brain that emits the observations
  - Learning how the observations were emitted from the brain states, and how the brain transitions from one underlying state to another

Latent variable models takeaways

- Used to analyze sequential data (such as time series)
- Hidden Markov model is an example of such a model, where the hidden states can take discrete values
- We can use HMMs for 3 main purposes:
  - Testing a hypothesis about underlying processes in the brain
  - Inferring the best sequence of underlying processes in the brain that emits the observations
  - Learning how the observations were emitted from the brain states, and how the brain transitions from one underlying state to another
- SSM (state-space model) is another type of latent variable model that accounts for continuous hidden states

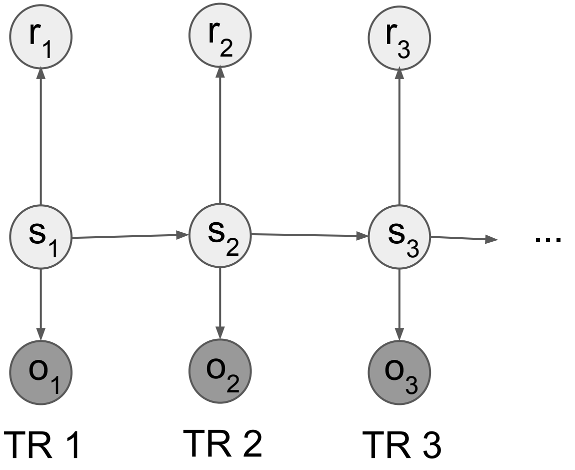
Can we model human learning with an HMM?

- To learn, we need some teaching, or reward, signal



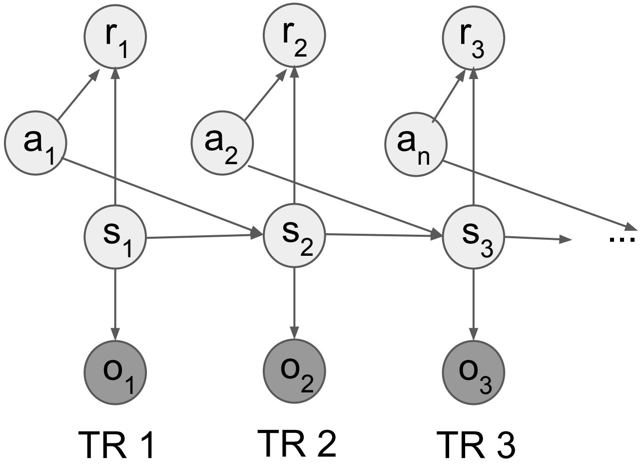
Can we model human learning with an HMM?

- To learn, we need some teaching, or reward, signal
- What if we incorporate a notion of reward in the HMM?



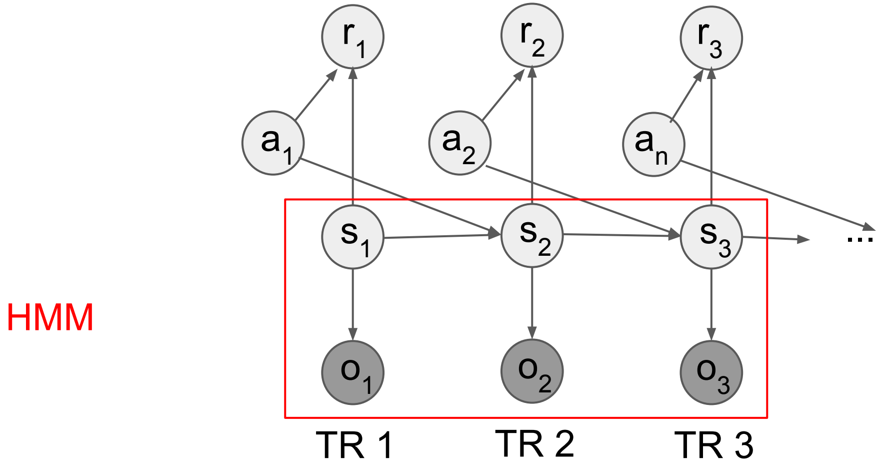
Can we model human learning with an HMM?

- To learn, we need some teaching, or reward, signal
- What if we incorporate a notion of reward in the HMM?
- Now, we incorporate actions we can take to transition to other states in order to maximize our reward



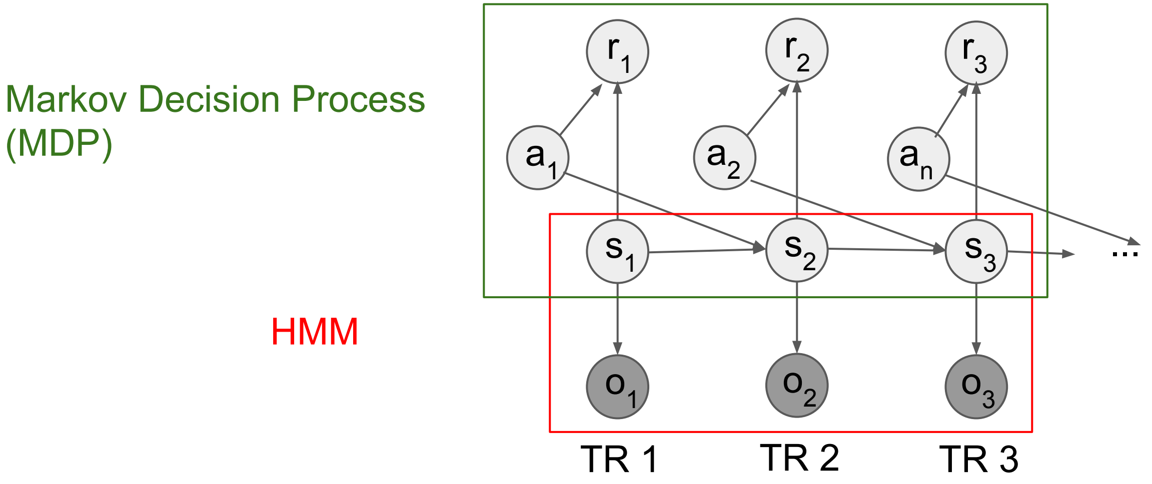
Can we model human learning with an HMM?

- To learn, we need some teaching, or reward, signal
- What if we incorporate a notion of reward in the HMM?
- Now, we incorporate actions we can take to transition to other states in order to maximize our reward



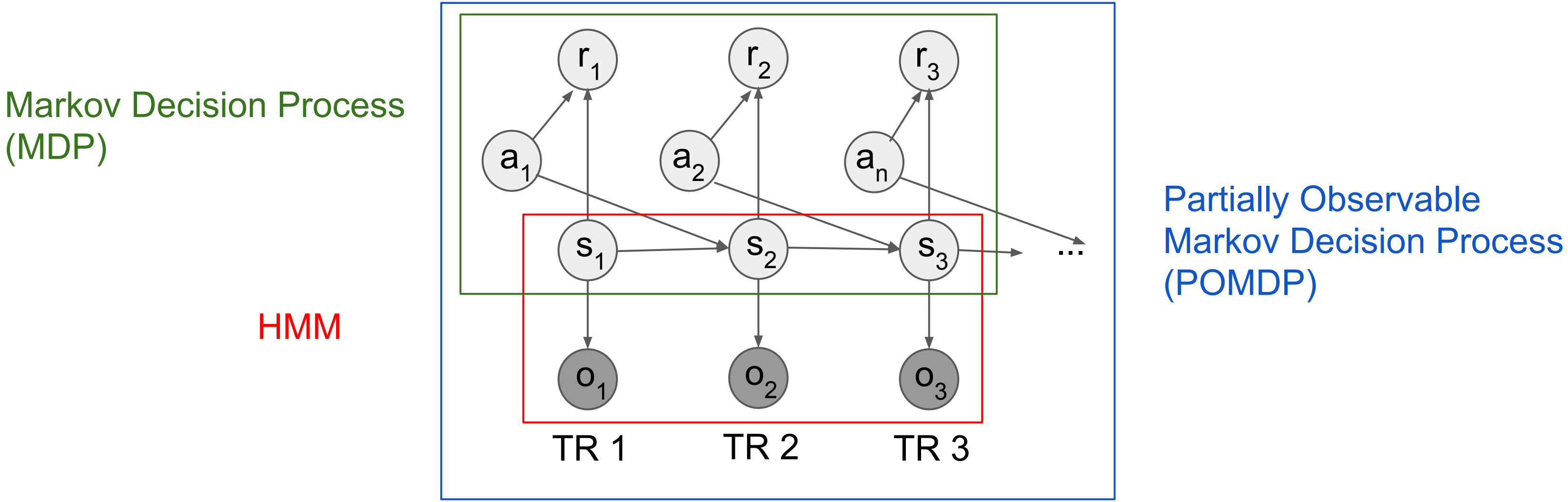
Can we model human learning with an HMM?

- To learn, we need some teaching, or reward, signal
- What if we incorporate a notion of reward in the HMM?
- Now, we incorporate actions we can take to transition to other states in order to maximize our reward



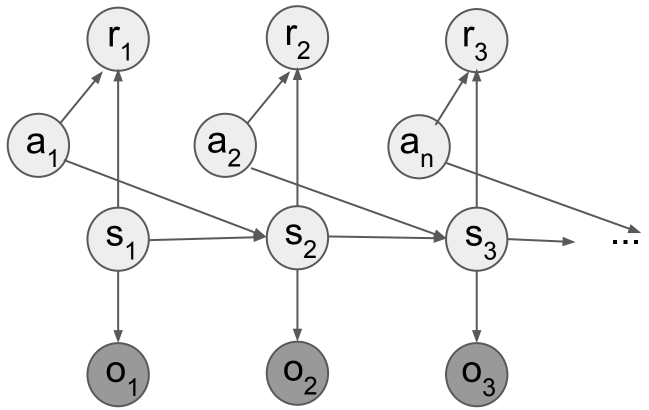
Can we model human learning with an HMM?

- To learn, we need some teaching, or reward, signal
- What if we incorporate a notion of reward in the HMM?
- Now, we incorporate actions we can take to transition to other states in order to maximize our reward



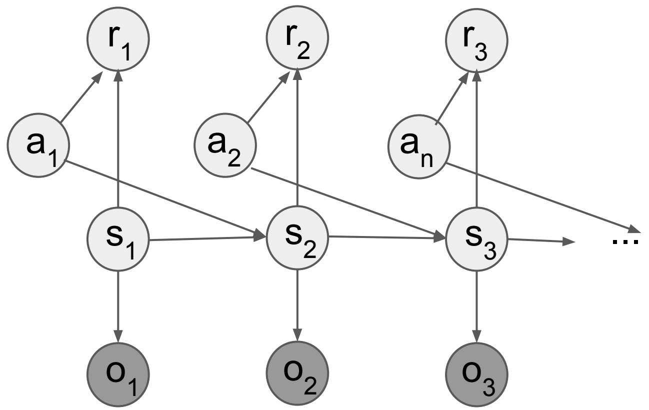
Reinforcement learning (RL): learning how to take the best actions to maximize expected future reward

- ❑ Delayed reward vs immediate reward



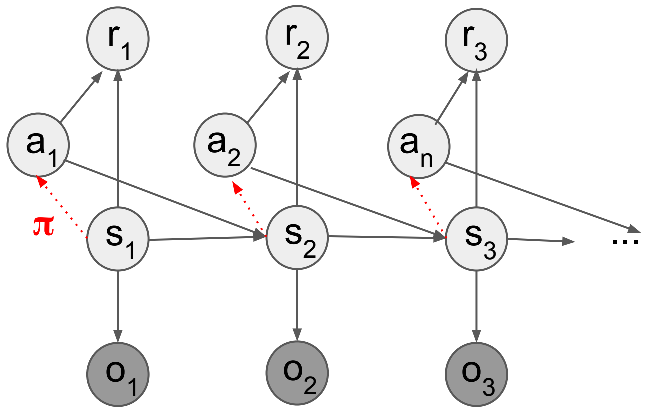
Reinforcement learning (RL): learning how to take the best actions to maximize expected future reward

- Delayed reward vs immediate reward
  - We can’t choose the best action to take right now with regular supervised methods



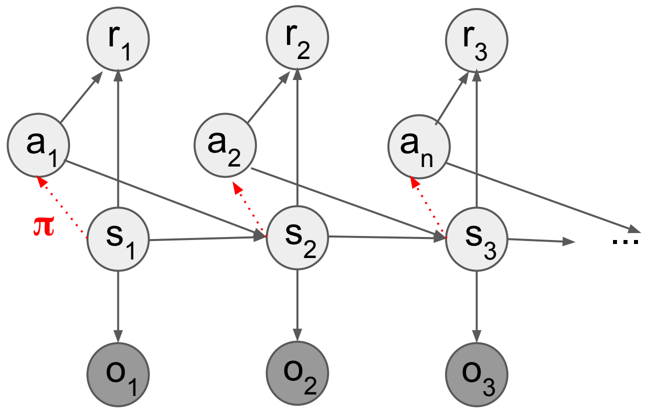
Reinforcement learning (RL): learning how to take the best actions to maximize expected future reward

- Delayed reward vs immediate reward
  - We can't choose the best action to take right now with regular supervised methods
- RL in a nutshell:
  - Learning function π: S->A that tells us what the best action to take is in the current state



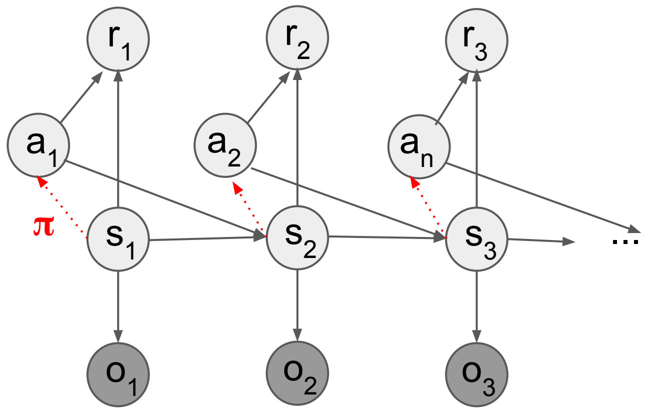
Reinforcement learning (RL): learning how to take the best actions to maximize expected future reward

- Delayed reward vs immediate reward
  - We can't choose the best action to take right now with regular supervised methods
- RL in a nutshell:
  - Learning function π: S->A that tells us what the best action to take is in the current state
    - π is also called a "policy"



Reinforcement learning (RL): learning how to take the best actions to maximize expected future reward

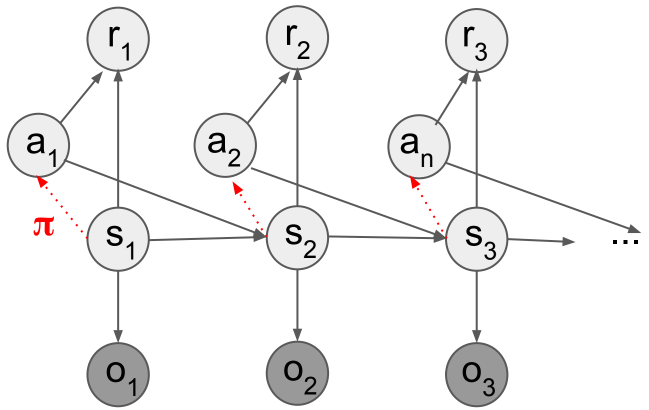
- Delayed reward vs immediate reward
  - We can't choose the best action to take right now with regular supervised methods
- RL in a nutshell:
  - Learning function π: S->A that tells us what the best action to take is in the current state
    - π is also called a "policy"
  - Given any policy π, we can define the expected future reward if we start at state s as:



Reinforcement learning (RL): learning how to take the best actions to maximize expected future reward

- Delayed reward vs immediate reward
  - We can't choose the best action to take right now with regular supervised methods
- RL in a nutshell:
  - Learning function π: S->A that tells us what the best action to take is in the current state
    - π is also called a "policy"
  - Given any policy π, we can define the expected future reward if we start at state s as:

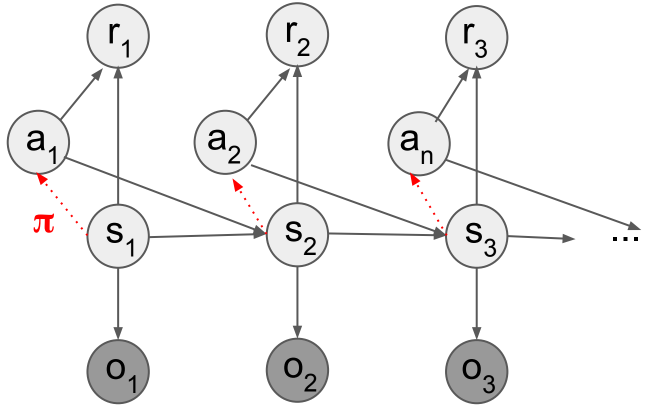
$$V^{\pi}(\boxed{s}) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$



Reinforcement learning (RL): learning how to take the best actions to maximize expected future reward

- Delayed reward vs immediate reward
  - We can't choose the best action to take right now with regular supervised methods
- RL in a nutshell:
  - Learning function π: S->A that tells us what the best action to take is in the current state
    - π is also called a "policy"
  - Given any policy π, we can define the expected future reward if we start at state s as:

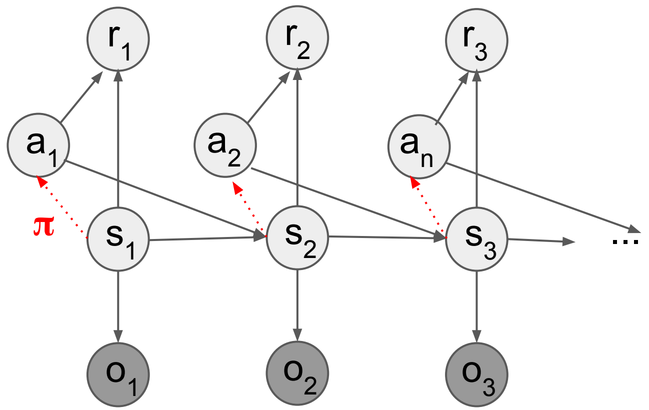
$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t \boxed{r_t}]$$



Reinforcement learning (RL): learning how to take the best actions to maximize expected future reward

- Delayed reward vs immediate reward
  - We can't choose the best action to take right now with regular supervised methods
- RL in a nutshell:
  - Learning function π: S->A that tells us what the best action to take is in the current state
    - π is also called a "policy"
  - Given any policy π, we can define the expected future reward if we start at state s as:

$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \boxed{\gamma^t} r_t]$$

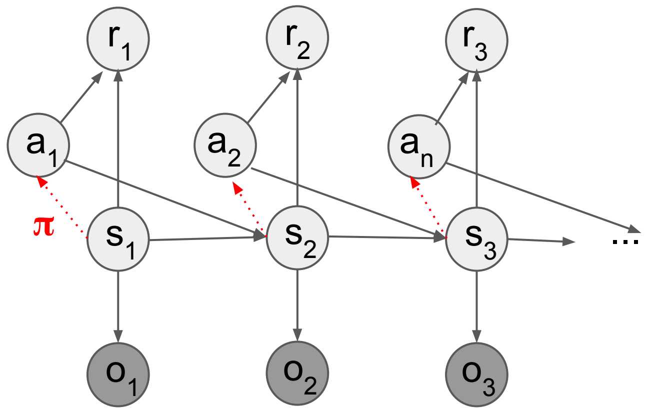


Reinforcement learning (RL): learning how to take the best actions to maximize expected future reward

- Delayed reward vs immediate reward
  - We can't choose the best action to take right now with regular supervised methods
- RL in a nutshell:
  - Learning function π: S->A that tells us what the best action to take is in the current state
    - π is also called a "policy"
  - Given any policy π, we can define the expected future reward if we start at state s as:

$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \boxed{\gamma^t} r_t]$$

  - We call this the value function of policy π

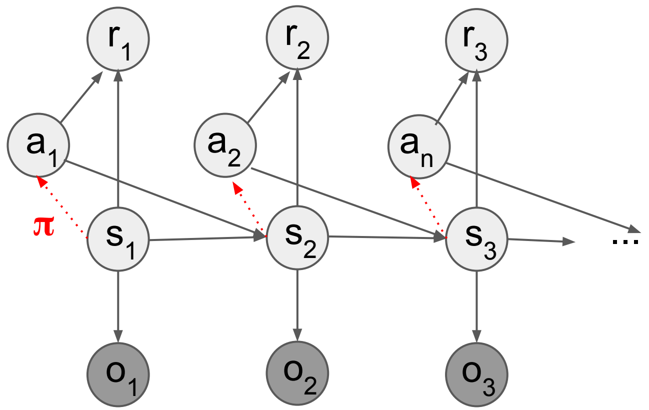


Reinforcement learning (RL): learning how to take the best actions to maximize expected future reward

- Delayed reward vs immediate reward
  - We can’t choose the best action to take right now with regular supervised methods
- RL in a nutshell:
  - Learning function π: S->A that tells us what the best action to take is in the current state
    - π is also called a “policy”
  - Given any policy π, we can define the expected future reward if we start at state s as:

$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \boxed{\gamma^t} r_t]$$

  - We call this the value function of policy π
  - Now, we can compute the best policy:



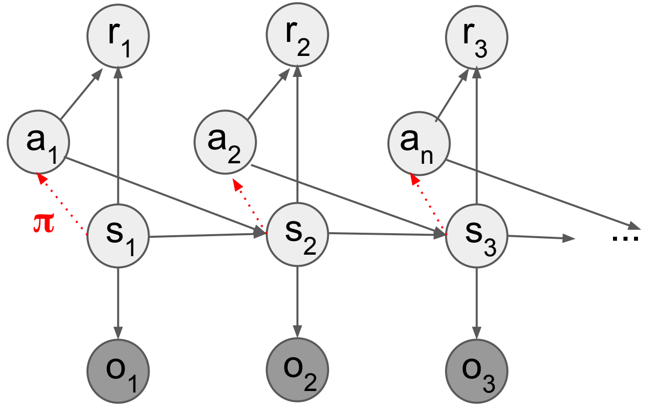
Reinforcement learning (RL): learning how to take the best actions to maximize expected future reward

- Delayed reward vs immediate reward
  - We can't choose the best action to take right now with regular supervised methods
- RL in a nutshell:
  - Learning function π: S->A that tells us what the best action to take is in the current state
    - π is also called a "policy"
  - Given any policy π, we can define the expected future reward if we start at state s as:

$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \boxed{\gamma^t} r_t]$$

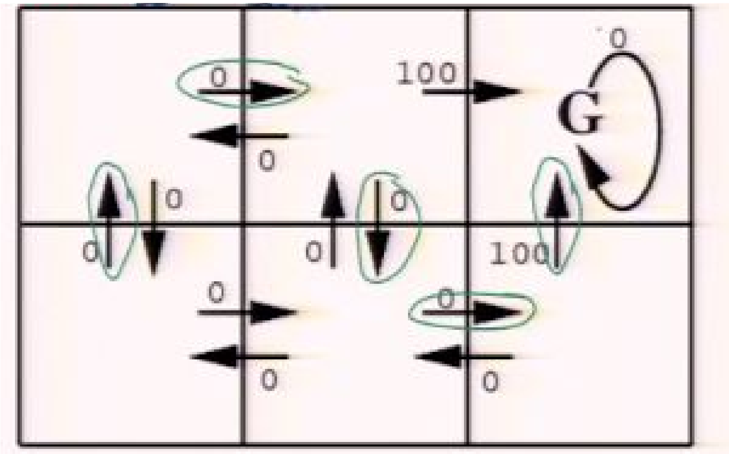
  - We call this the value function of policy π
  - Now, we can compute the best policy:

$$\pi^* = \arg\max_{\pi} V^{\pi}(s), \quad (\forall s)$$



Example: what are the $V^{\pi}(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy **π**
- Let γ = 0.9

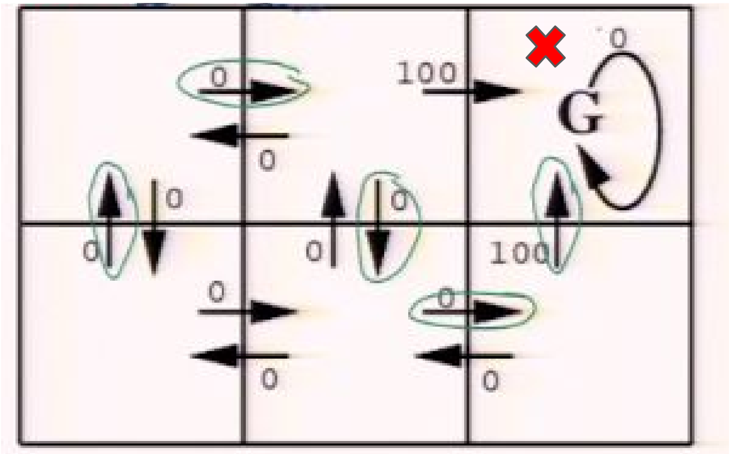


$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the $V^{\pi}(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy π
- Let $\gamma = 0.9$

What is $V^{\pi}(\mathbf{x})$?

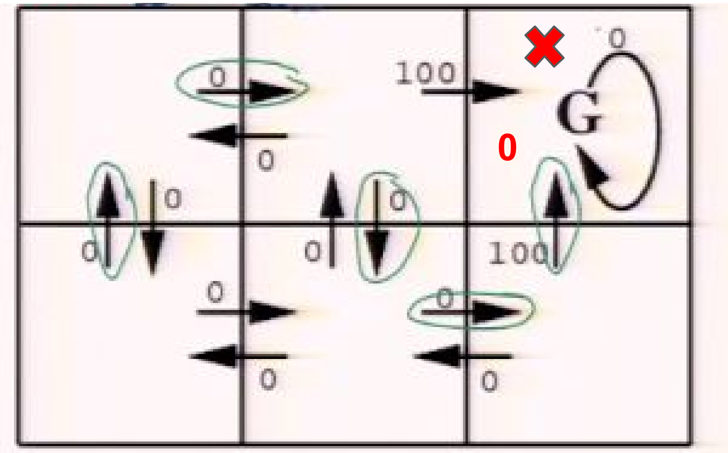


$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the $V^\pi(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy π
- Let γ = 0.9

What is V^π(x)? 0

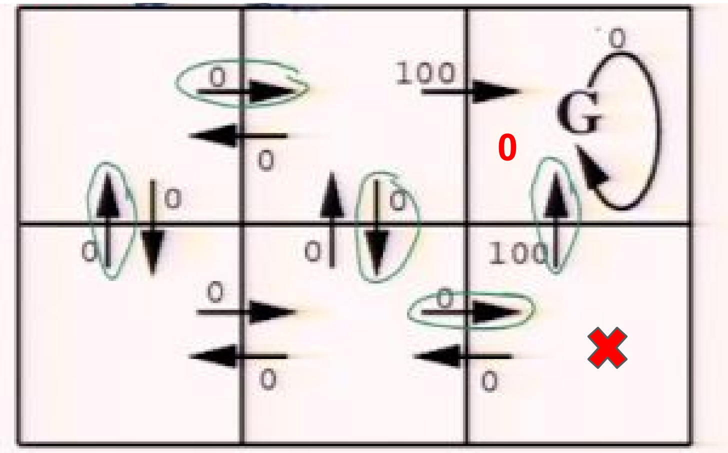


$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the $V^\pi(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy **π**
- Let γ = 0.9

What is V^π(**x**)?

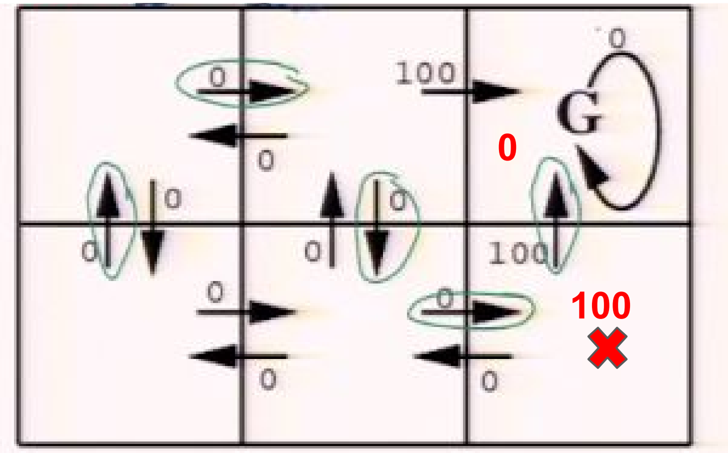


$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the $V^{\pi}(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy π
- Let $\gamma = 0.9$

What is V^{π}(x)? 100

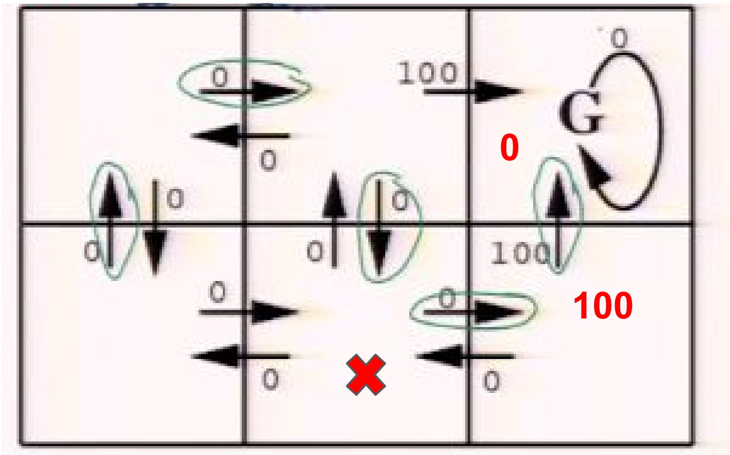


$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the $V^{\pi}(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy π
- Let $\gamma = 0.9$

What is $V^{\pi}(x)$?

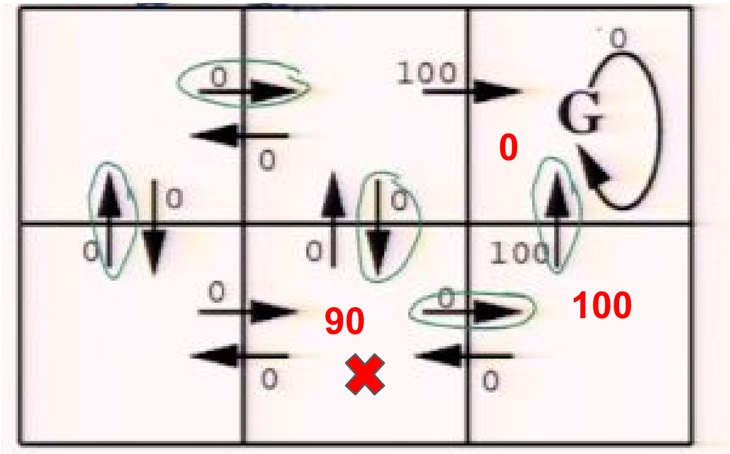


$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the $V^{\pi}(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy π
- Let γ = 0.9

What is V^{π}(**x**)? 0.9 x100 = 90

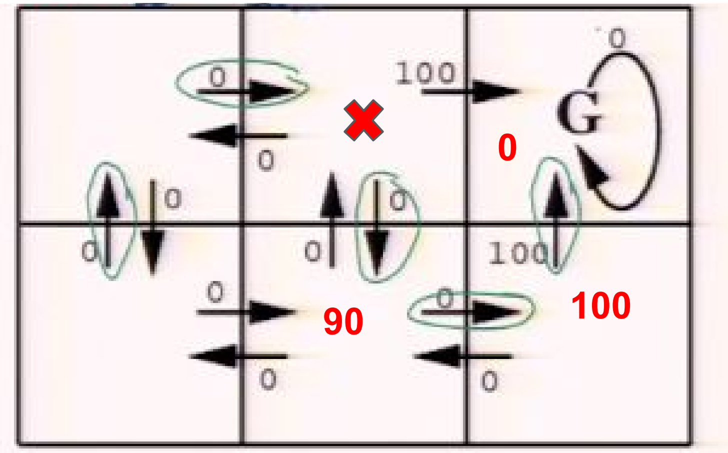


$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the $V^{\pi}(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy π
- Let γ = 0.9

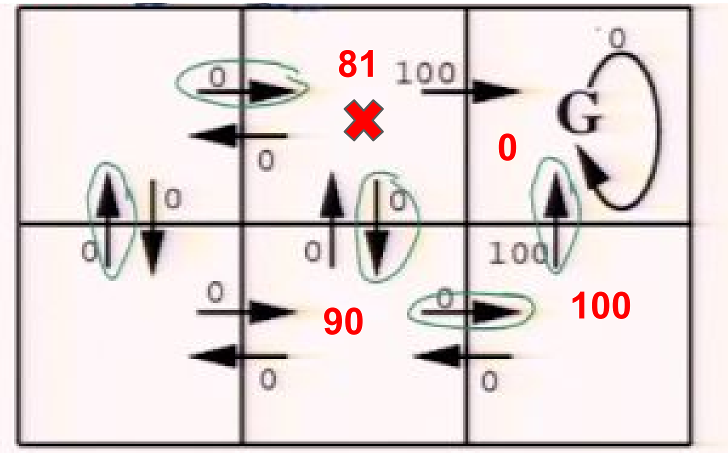
What is V^{π}(x)?



$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the $V^{\pi}(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy π
- Let γ = 0.9



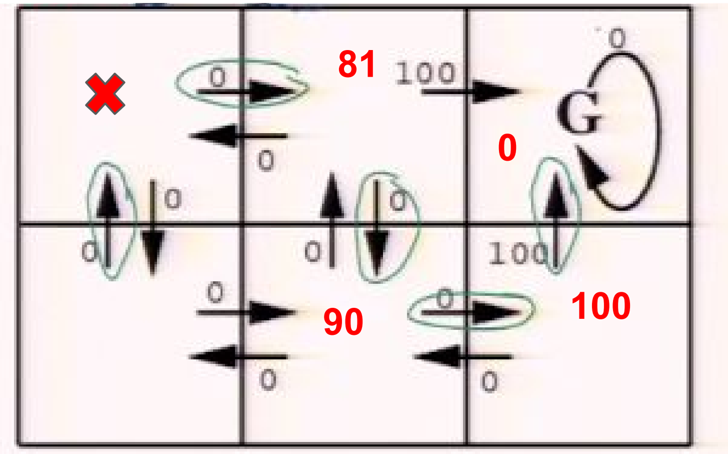
What is V^{π}(x)? 0.9x0.
9x100 = 81

$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the $V^{\pi}(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy π
- Let $\gamma = 0.9$

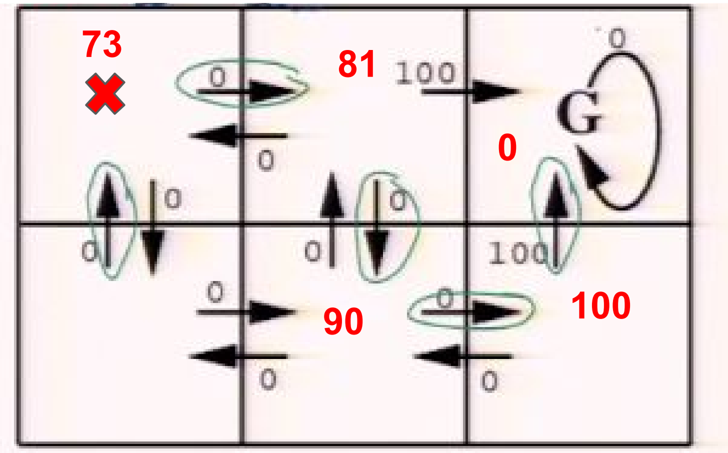
What is $V^{\pi}(x)$?



$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the $V^{\pi}(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy π
- Let $\gamma = 0.9$



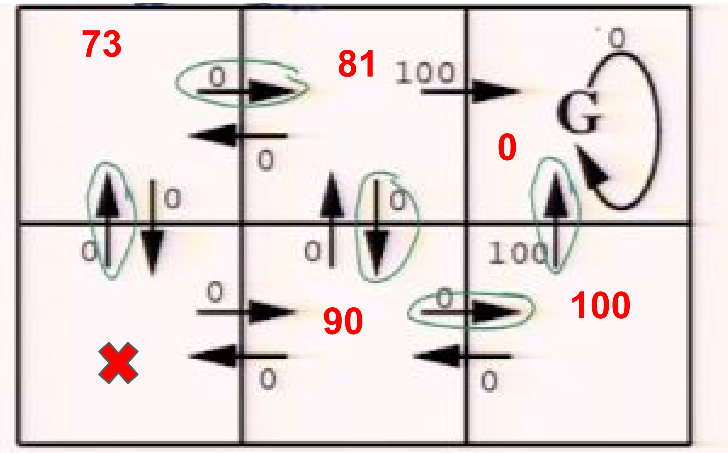
What is $V^{\pi}(x)$? 0.9x0.9x0.9x100 = 73

$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the $V^\pi(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy π
- Let $\gamma = 0.9$

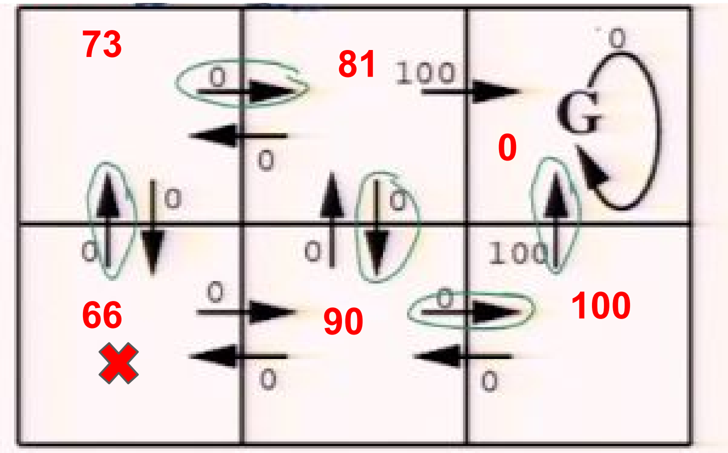
What is $V^\pi(x)$?



$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the $V^\pi(s)$ values?

- Image there is a robot that needs to get to the goal state G
- Each square is a state
- Circled arrows show the policy π
- Let γ = 0.9

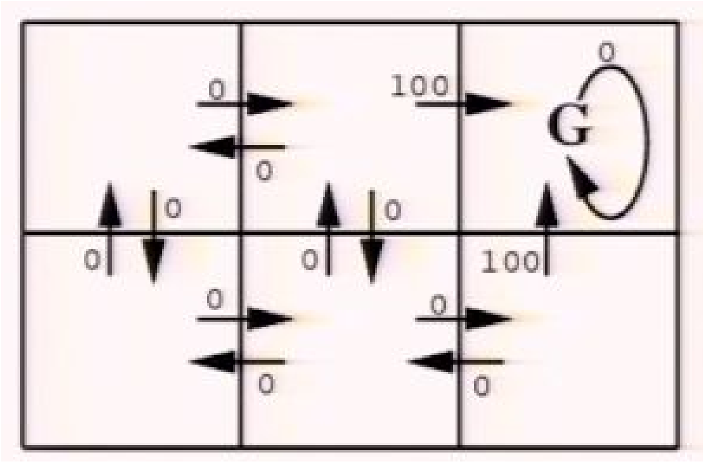


What is V^π(**x**)? 0.9x0.9x0.9x100 = 66

$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

- ❑ What is the best policy?

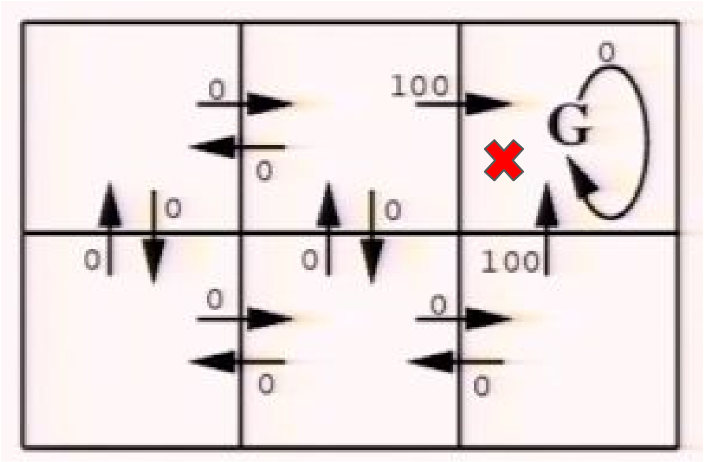


$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

- What is the best policy?

What is $V^*(\textbf{x})$?

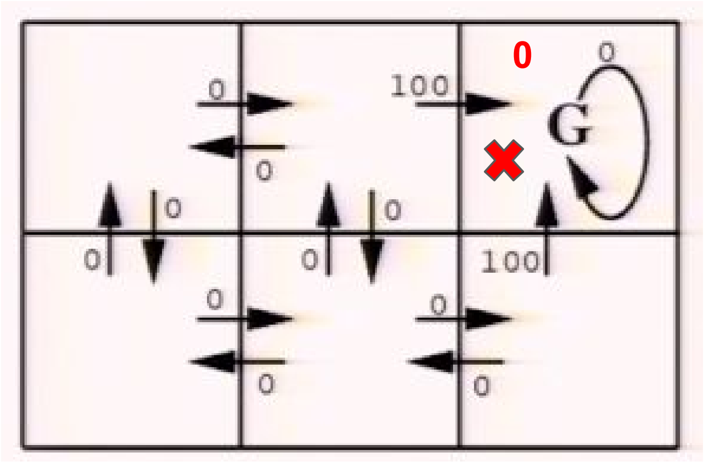


$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

- What is the best policy?

What is V^*(**x**)? Still 0

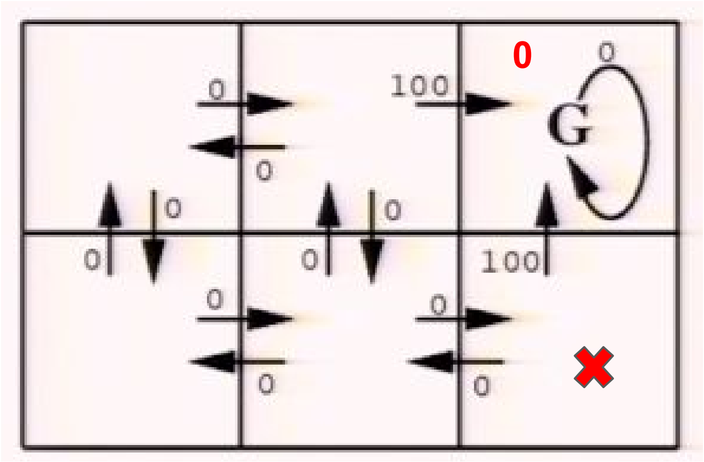


$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

- What is the best policy?

What is $V^*(\mathbf{x})$?

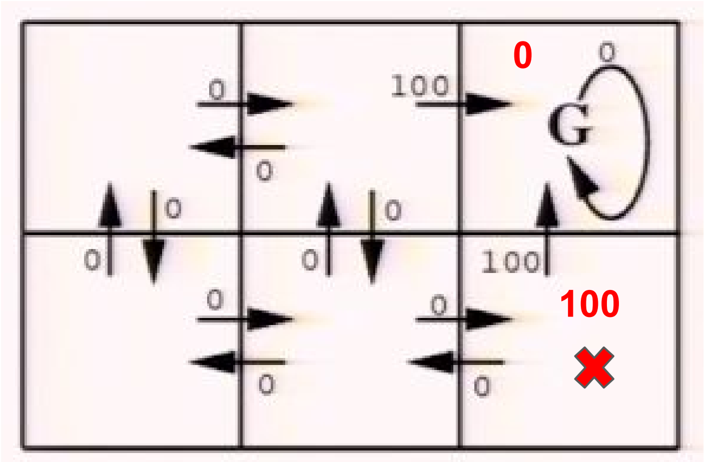


$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

- What is the best policy?

What is $V^*($**x**$)$? 100

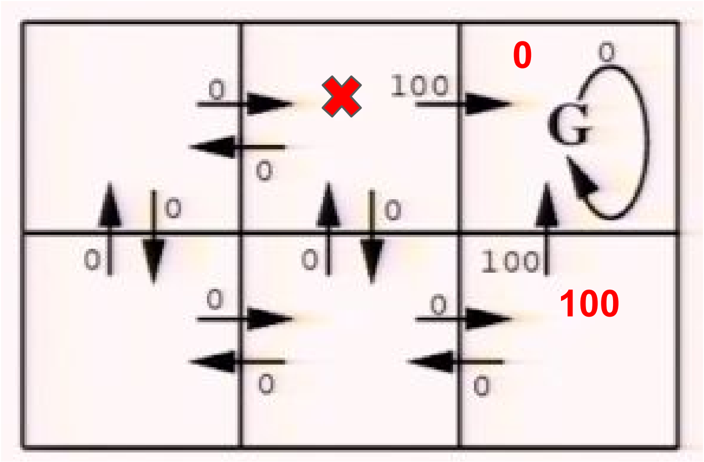


$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

- What is the best policy?

What is $V^*(\mathbf{x})$?

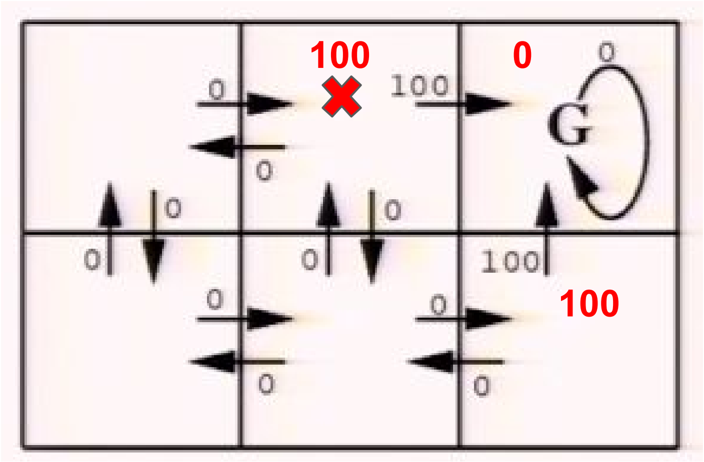


$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

❑ What is the best policy?

What is V^*(**x**)? 100

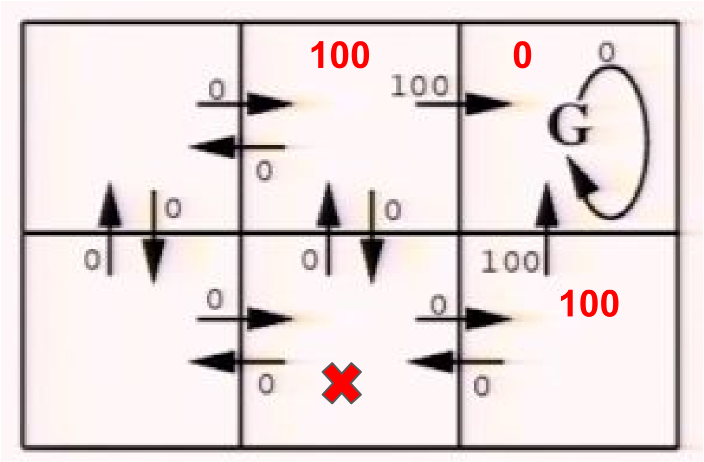


$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

- ❑ What is the best policy?

What is $V^*(\mathbf{x})$?

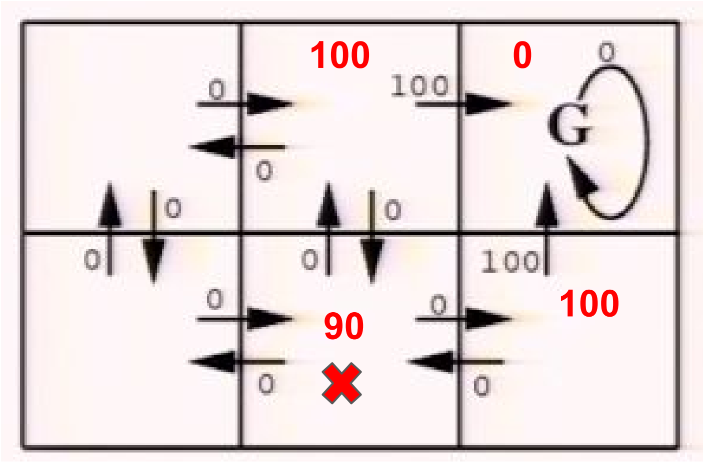


$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

- What is the best policy?

What is $V^*(\mathbf{x})$? 90

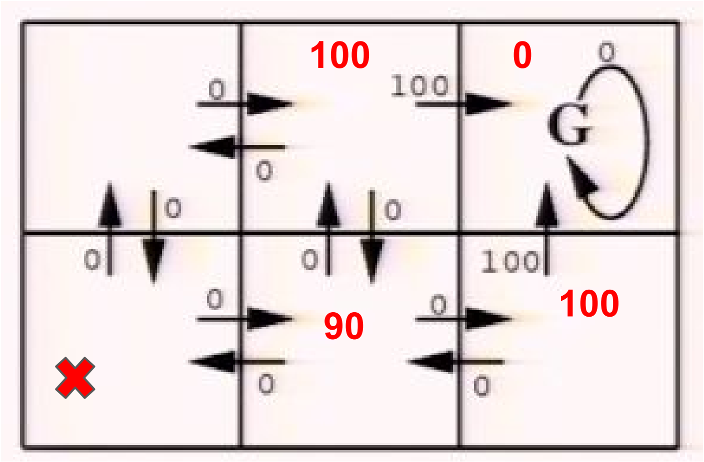


$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

- What is the best policy?

What is $V^*(\mathbf{x})$?

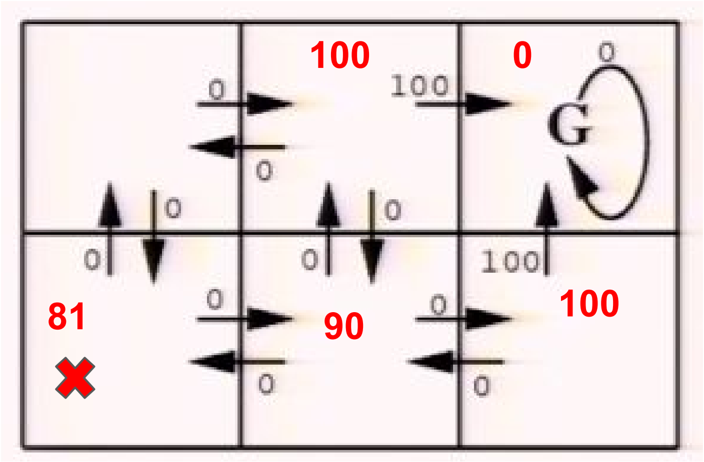


$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

- What is the best policy?

What is $V^*(\mathbf{x})$? 81

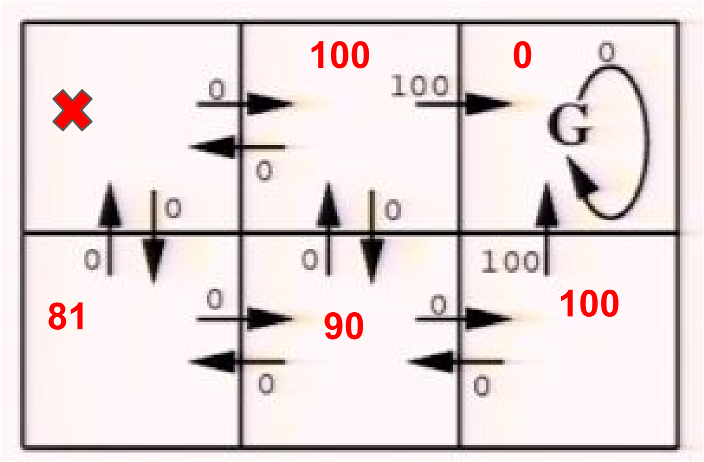


$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

- ❑ What is the best policy?

What is $V^*(\mathbf{x})$?

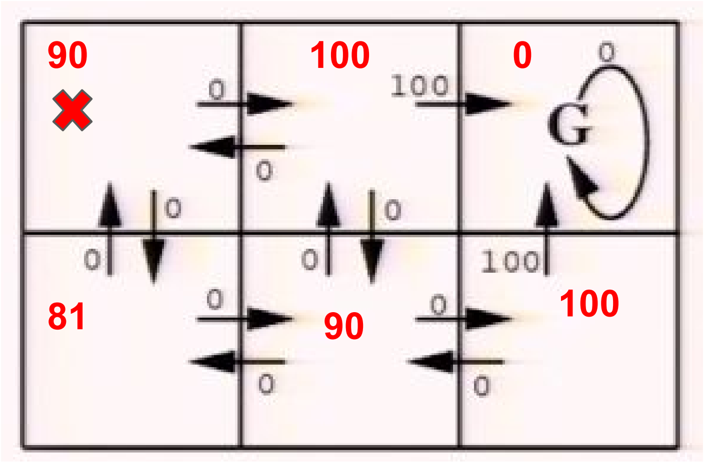


$$V^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

Example: what are the best $V^*(s)$ values?

- ❏ What is the best policy?

What is $V^*(\mathbf{x})$? 90



$$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$$

RL takeaways

- Reinforcement learning introduces the notion of delayed reward and uses it to learn the best actions that will maximize this future reward

RL takeaways

- Reinforcement learning introduces the notion of delayed reward and uses it to learn the best actions that will maximize this future reward
- Two ways to learn the best policy:

RL takeaways

- Reinforcement learning introduces the notion of delayed reward and uses it to learn the best actions that will maximize this future reward
- Two ways to learn the best policy:
  - Q-learning
  - Temporal Difference (TD) learning

RL takeaways

- Reinforcement learning introduces the notion of delayed reward and uses it to learn the best actions that will maximize this future reward
- Two ways to learn the best policy:
  - Q-learning
  - Temporal Difference (TD) learning
    - Error in value prediction using TD-learning is the best computational model of dopamine release in monkeys during anticipation of future reward => hypothesis that dopamine does reward prediction (Schultz et al. Science, 1997)

RL takeaways

- Reinforcement learning introduces the notion of delayed reward and uses it to learn the best actions that will maximize this future reward
- Two ways to learn the best policy:
  - Q-learning
  - Temporal Difference (TD) learning
    - Error in value prediction using TD-learning is the best computational model of dopamine release in monkeys during anticipation of future reward => hypothesis that dopamine does reward prediction (Schultz et al. Science, 1997)
- Recent advances in RL: combining it with deep learning!

RL takeaways

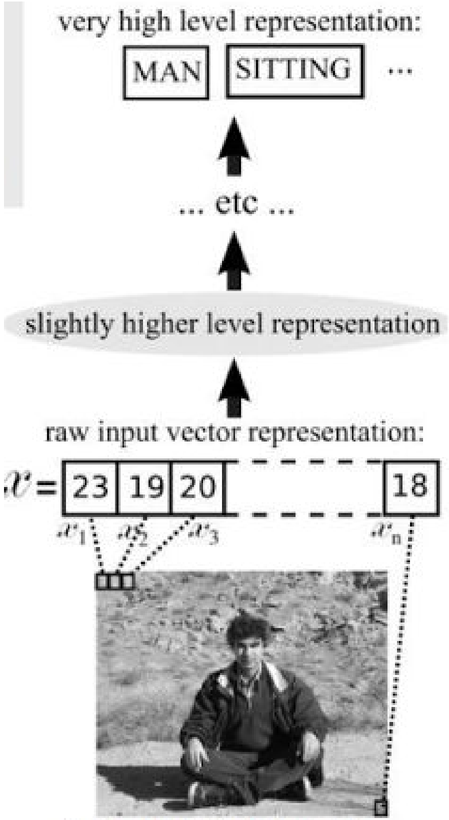
- Reinforcement learning introduces the notion of delayed reward and uses it to learn the best actions that will maximize this future reward
- Two ways to learn the best policy:
  - Q-learning
  - Temporal Difference (TD) learning
    - Error in value prediction using TD-learning is the best computational model of dopamine release in monkeys during anticipation of future reward => hypothesis that dopamine does reward prediction (Schultz et al. Science, 1997)
- Recent advances in RL: combining it with deep learning!
  - learning deep policy and value networks

Deep learning!

- Loosely inspired by neural connections in the brain

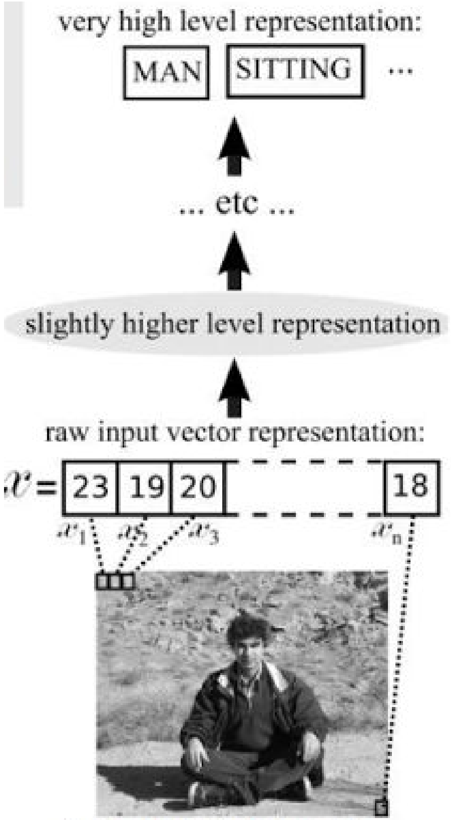
Deep learning!

- Loosely inspired by neural connections in the brain
- Goal: learn hierarchy of features
  - higher level features are formed by lower level features



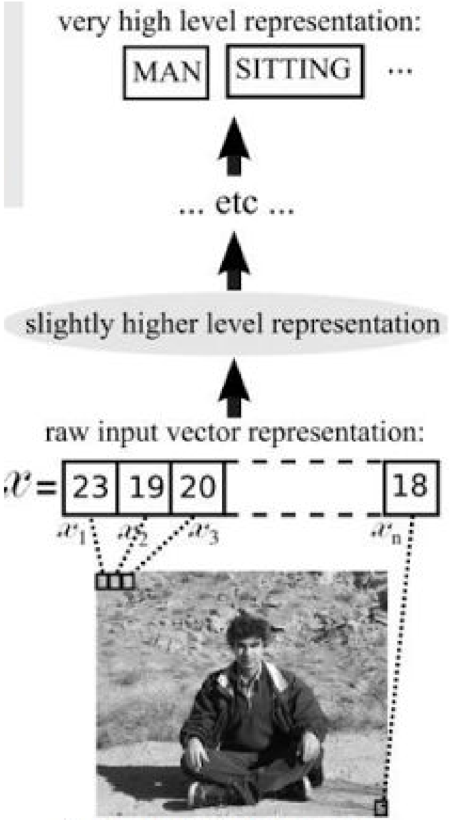
Deep learning!

- Loosely inspired by neural connections in the brain
- Goal: learn hierarchy of features
    - higher level features are formed by lower level features
- Used in neuroscience to study intermediate stages of representations



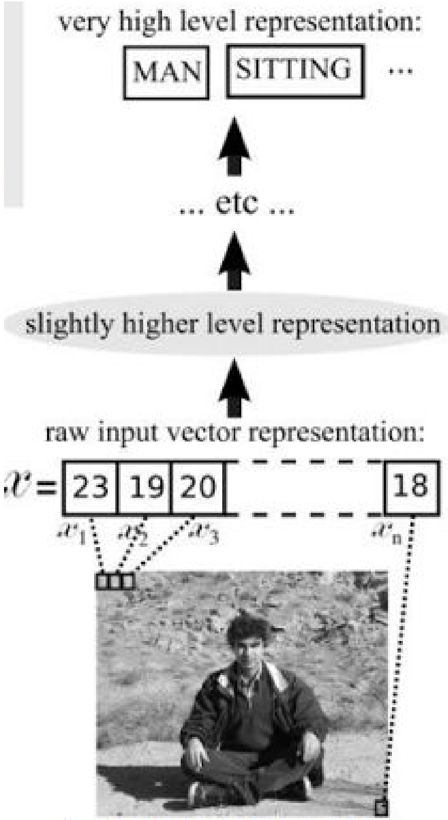
Deep learning!

- Loosely inspired by neural connections in the brain
- Goal: learn hierarchy of features
  - higher level features are formed by lower level features
- Used in neuroscience to study intermediate stages of representations
- There are a few important types:

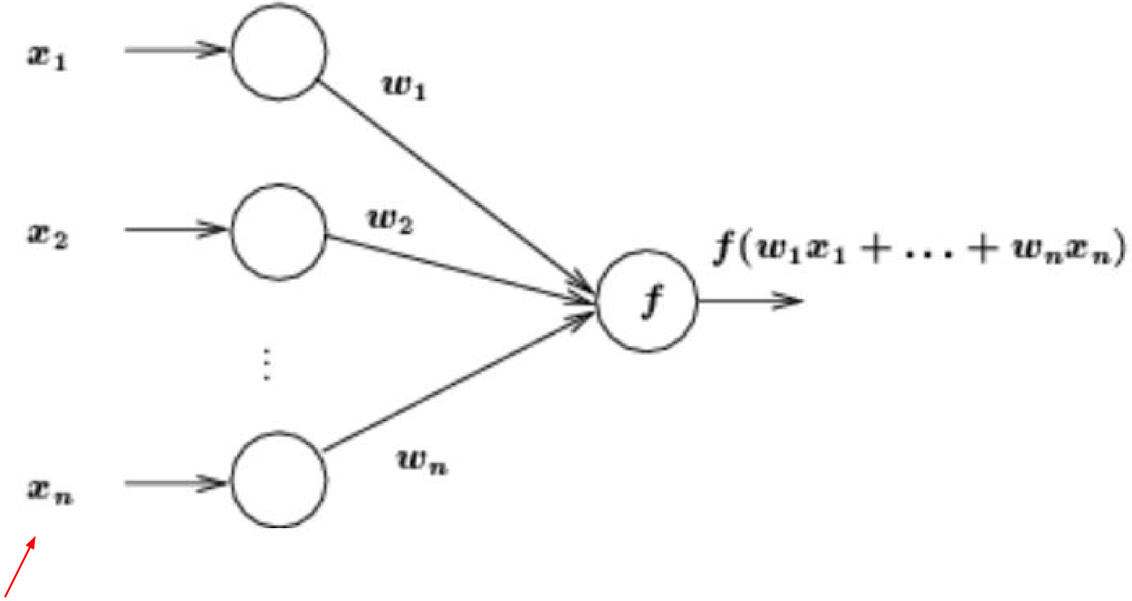


Deep learning!

- Loosely inspired by neural connections in the brain
- Goal: learn hierarchy of features
  - higher level features are formed by lower level features
- Used in neuroscience to study intermediate stages of representations
- There are a few important types:
  - Neural networks
  - Recurrent neural networks (RNNs)
    - Long short-term memory network (LSTM)
  - Deep belief networks (DBNs)
  - Convolutional neural networks (CNNs)

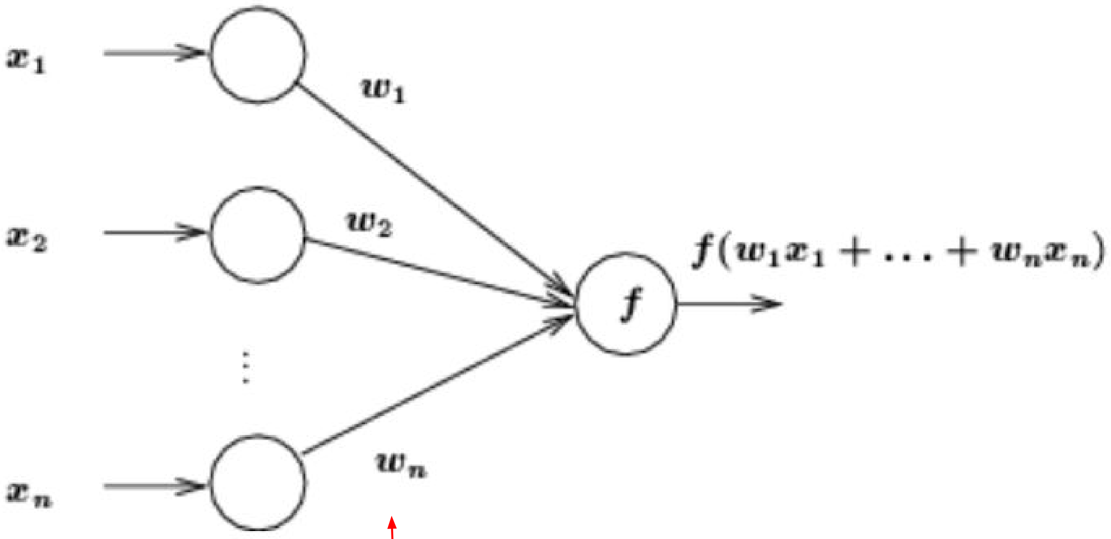


Artificial neural networks are inspired by connections between real neurons



data instance (like dendritic input)

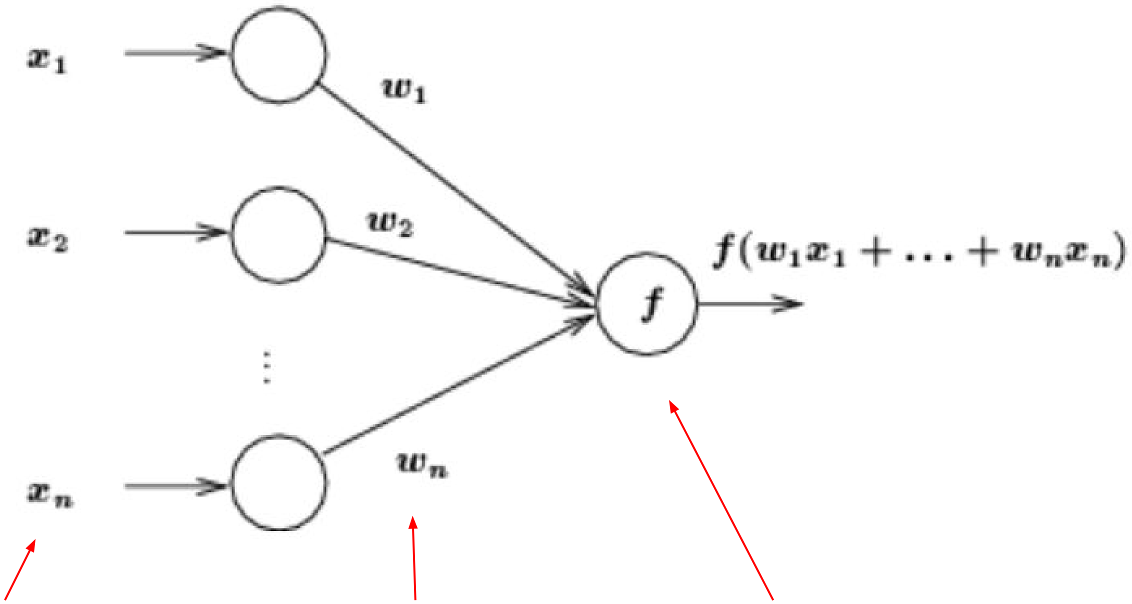
Artificial neural networks are inspired by connections between real neurons



data instance (like dendritic input)

learned weights (like synaptic weights)

Artificial neural networks are inspired by connections between real neurons

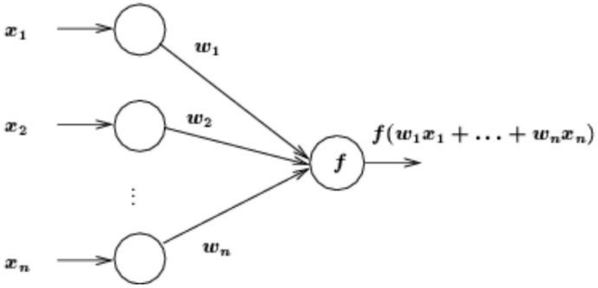


data instance (like dendritic input)

learned weights (like synaptic weights)

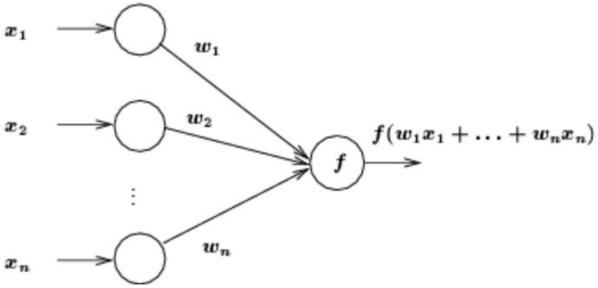
activation function (like firing threshold in neurons)

What is a good activation function f?



- Want non-linear f

What is a good activation function f?



- Want non-linear f
- Some typical examples:
  - logistic (or sigmoid) function

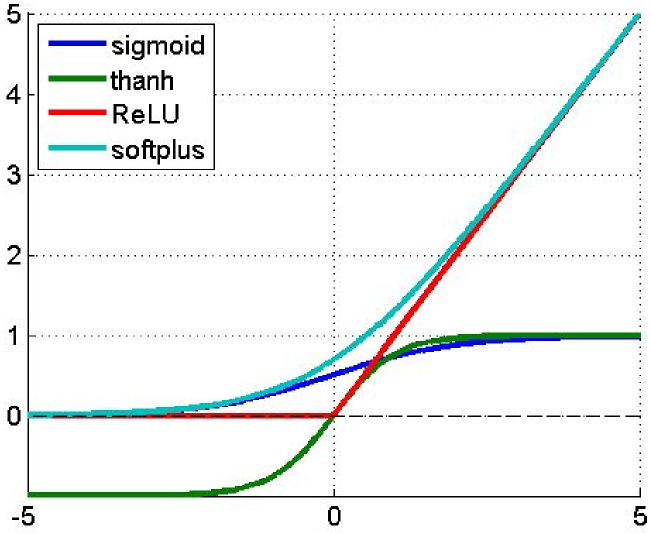
    $$f(x) = (1 + e^{-x})^{-1}$$

  - hyperbolic tangent function

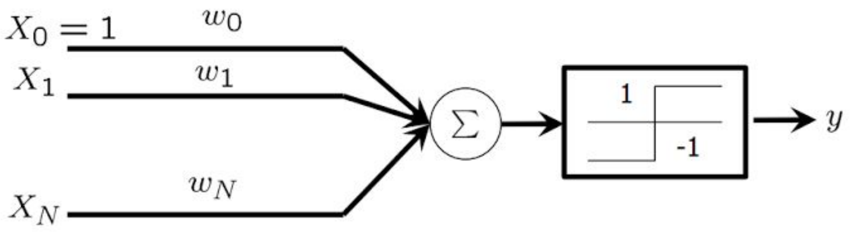
    $$f(x) = \tanh(x)$$

  - rectified linear function

    $$f(x) = \max(0, x)$$



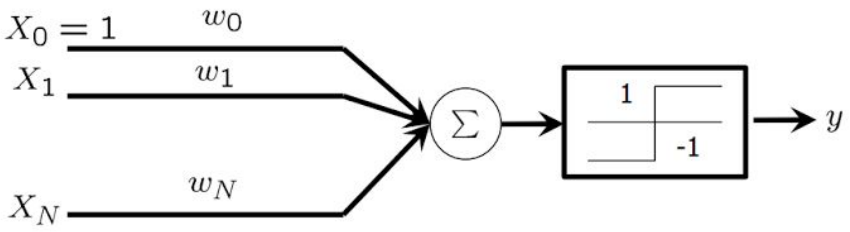
Perceptron: simplest neural network



$$y = sgn(\mathbf{w}^T \mathbf{x})$$

- activation function is just a threshold at 0 (checks for the sign of the weighted input sum)

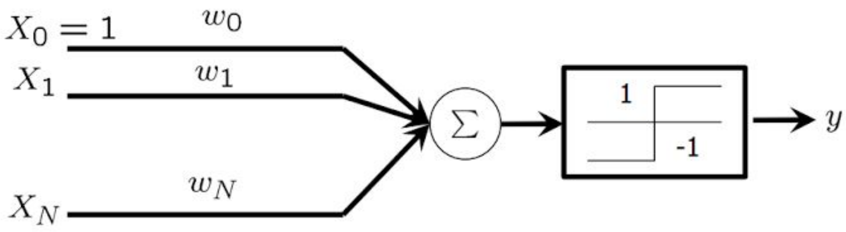
Perceptron: simplest neural network



$$y = sgn(\mathbf{w}^T\mathbf{x})$$

- activation function is just a threshold at 0 (checks for the sign of the weighted input sum)
- no hidden layer

Perceptron: simplest neural network

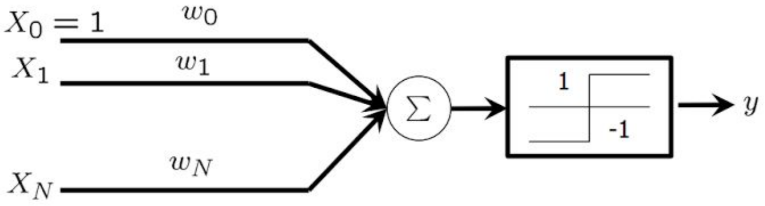


$$y = sgn(\mathbf{w}^T \mathbf{x})$$

- activation function is just a threshold at 0 (checks for the sign of the weighted input sum)
- no hidden layer
- goal: given the input x and label y, find weights w such that y = sgn(w^Tx)

How do we learn the weights in the perceptron?

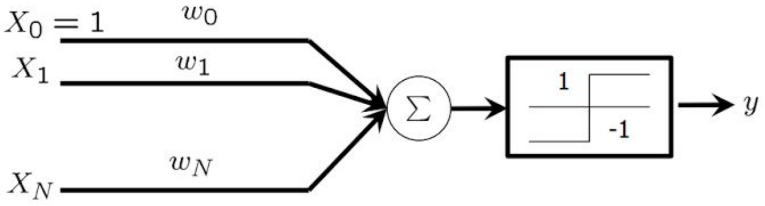
- Key idea: find classification error and update weights to reduce this error



$$y = sgn(\mathbf{w}^T \mathbf{x})$$

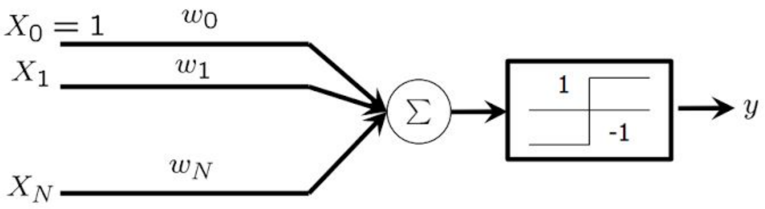
How do we learn the weights in the perceptron?

- Key idea: find classification error and update weights to reduce this error
- step 1: initialize $w_0,...,w_N$ randomly



$$y = sgn(\mathbf{w}^T\mathbf{x})$$

How do we learn the weights in the perceptron?

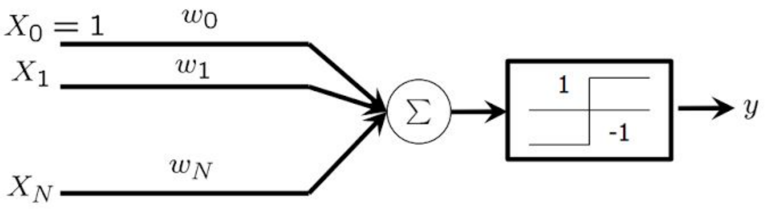


$$y = sgn(\mathbf{w}^T\mathbf{x})$$

- Key idea: find classification error and update weights to reduce this error
- step 1: initialize $w_0, ..., w_N$ randomly
- step 2: let x^k be a training data instance miscl w(t-1)

How do we learn the weights in the perceptron?

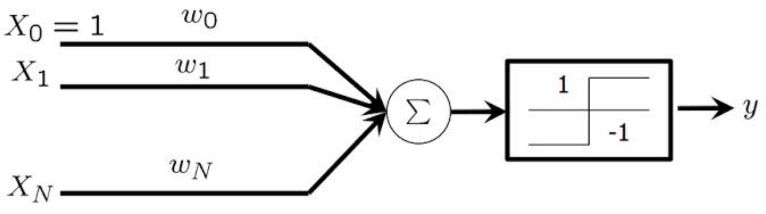
- Key idea: find classification error and update weights to reduce this error
- step 1: initialize $w_0,...,w_N$ randomly
- step 2: let x^k be a training data instance miscl
  w(t-1)
- step 3: if there is such an instance, calculate classification error $\varepsilon(t) = y^k - sgn(w(t-1)^T x^k)$



$$y = sgn(\mathbf{w}^T\mathbf{x})$$

How do we learn the weights in the perceptron?

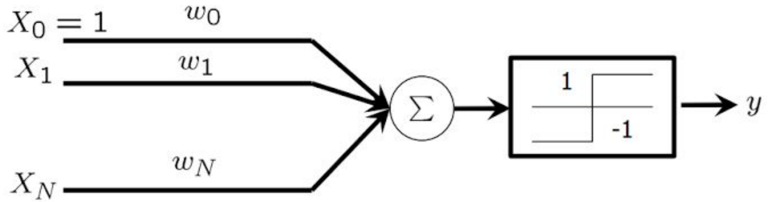
- Key idea: find classification error and update weights to reduce this error
- step 1: initialize $w_0,...,w_N$ randomly
- step 2: let x^k be a training data instance miscl
  w(t-1)
- step 3: if there is such an instance, calculate classification error $\varepsilon(t) = y^k - sgn(w(t-1)^T x^k)$
- step 4: account for error by updating weights:
  - $w(t) = w(t-1) + \alpha\varepsilon(t)x^k$



$$y = sgn(\mathbf{w}^T\mathbf{x})$$

How do we learn the weights in the perceptron?

- Key idea: find classification error and update weights to reduce this error

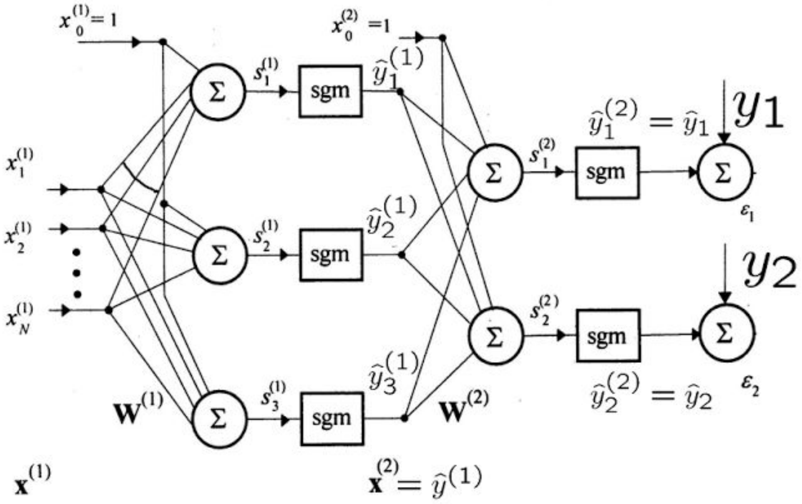


$$y = sgn(\mathbf{w}^T\mathbf{x})$$

- step 1: initialize $w_0,...,w_N$ randomly
- step 2: let x^k be a training data instance miscl w(t-1)
- step 3: if there is such an instance, calculate classification error $\varepsilon(t) = y^k - sgn(w(t-1)^T x^k)$
- step 4: account for error by updating weights:
  - $w(t) = w(t-1) + \alpha\varepsilon(t)x^k$
- step 5: t = t+1, end when there are no misclassifications

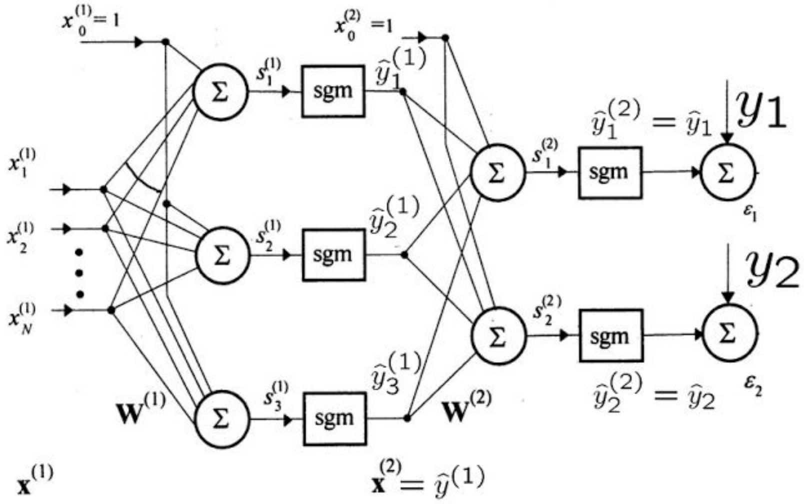
We add more layers => multilayer perceptron (MLP)

- Additional layers (not input or output) are called “hidden” layers



We add more layers => multilayer perceptron (MLP)

- Additional layers (not input or output) are called “hidden” layers



- Still need to learn weights W, but now we have multiple layers

Learning weights with multiple layers: backpropagation of error

- When there is a misclassification, different neurons have different amount of responsibility for it

Learning weights with multiple layers: backpropagation of error

- When there is a misclassification, different neurons have different amount of responsibility for it
- Key observation: the misclassification error for a neuron in layer m can be calculated from the misclassification errors for all of the neurons in layer m+1
    - because the network is fully connected

Learning weights with multiple layers: backpropagation of error

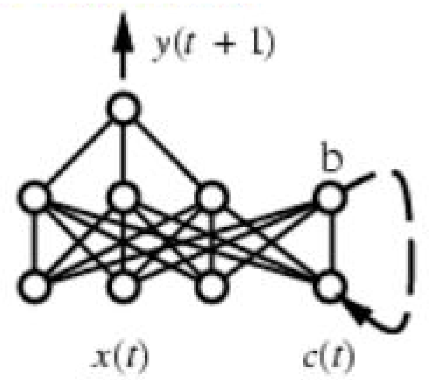
- When there is a misclassification, different neurons have different amount of responsibility for it
- Key observation: the misclassification error for a neuron in layer m can be calculated from the misclassification errors for all of the neurons in layer m+1
    - because the network is fully connected
- So we can calculate the final misclassification error and propagate is backward

Learning weights with multiple layers: backpropagation of error

- When there is a misclassification, different neurons have different amount of responsibility for it
- Key observation: the misclassification error for a neuron in layer m can be calculated from the misclassification errors for all of the neurons in layer m+1
  - because the network is fully connected
- So we can calculate the final misclassification error and propagate is backward
- Update weights according to the error

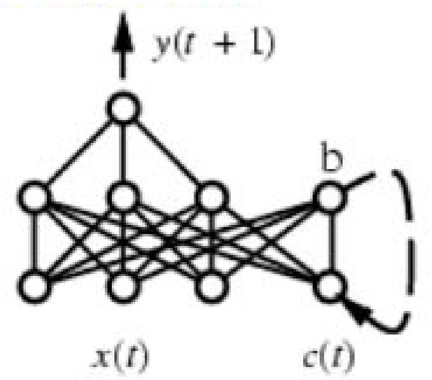
Neural networks for sequential data: recurrent NNs

- In sequential data, history can help predict the future



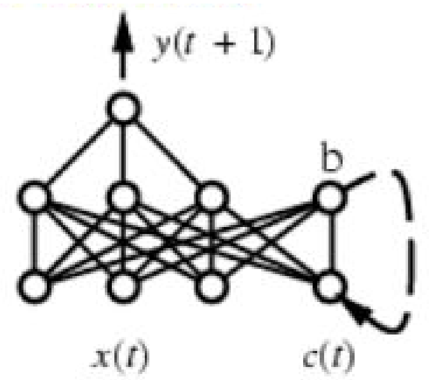
Neural networks for sequential data: recurrent NNs

- In sequential data, history can help predict the future
- Would like to explicitly capture state history in the neural network



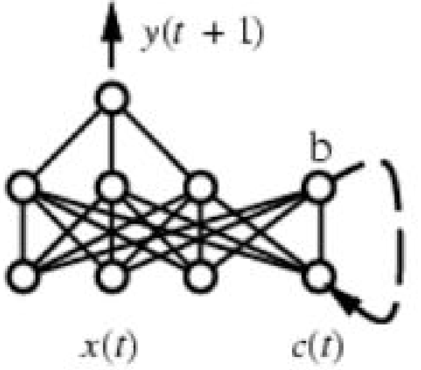
Neural networks for sequential data: recurrent NNs

- In sequential data, history can help predict the future
- Would like to explicitly capture state history in the neural network
    - Add a recurrent state



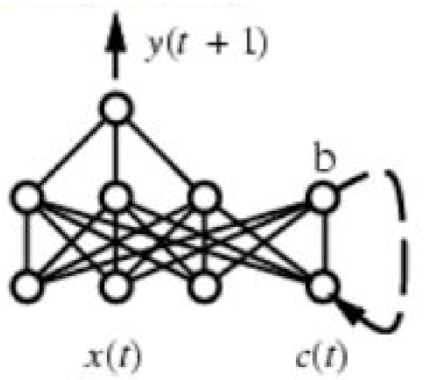
Neural networks for sequential data: recurrent NNs

- In sequential data, history can help predict the future
- Would like to explicitly capture state history in the neural network
  - Add a recurrent state
- In regular RNNs, the recurrent state is overwritten frequently



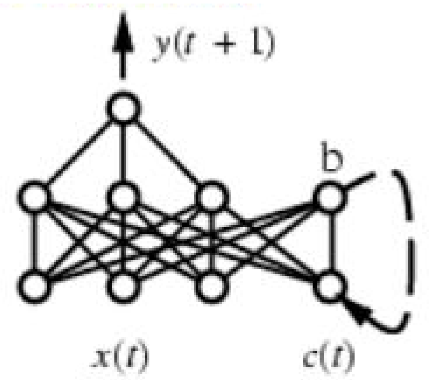
Neural networks for sequential data: recurrent NNs

- In sequential data, history can help predict the future
- Would like to explicitly capture state history in the neural network
  - Add a recurrent state
- In regular RNNs, the recurrent state is overwritten frequently
  - Good if important events happen close together



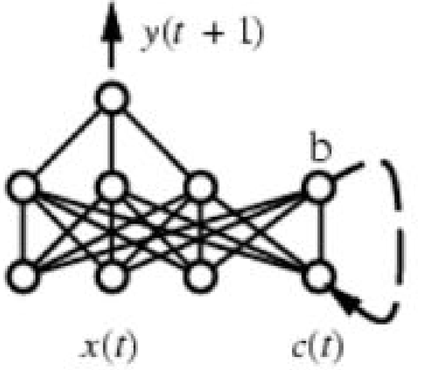
Neural networks for sequential data: recurrent NNs

- In sequential data, history can help predict the future
- Would like to explicitly capture state history in the neural network
  - Add a recurrent state
- In regular RNNs, the recurrent state is overwritten frequently
  - Good if important events happen close together
- But what if there are large delays before important events?



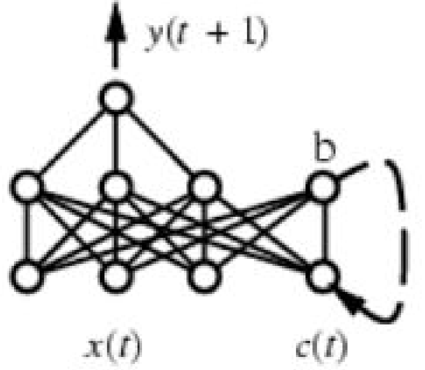
Neural networks for sequential data: recurrent NNs

- In sequential data, history can help predict the future
- Would like to explicitly capture state history in the neural network
  - Add a recurrent state
- In regular RNNs, the recurrent state is overwritten frequently
  - Good if important events happen close together
- But what if there are large delays before important events?
  - Long short-term memory (LSTM) networks!



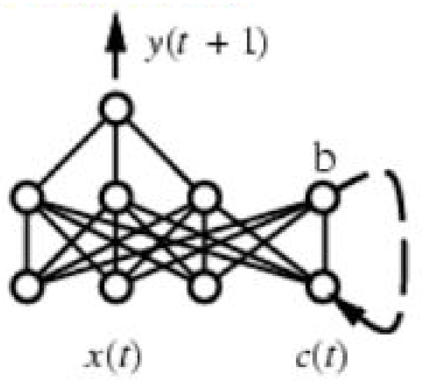
Neural networks for sequential data: recurrent NNs

- In sequential data, history can help predict the future
- Would like to explicitly capture state history in the neural network
  - Add a recurrent state
- In regular RNNs, the recurrent state is overwritten frequently
  - Good if important events happen close together
- But what if there are large delays before important events?
  - Long short-term memory (LSTM) networks!
    - Best known result in connected handwriting recognition



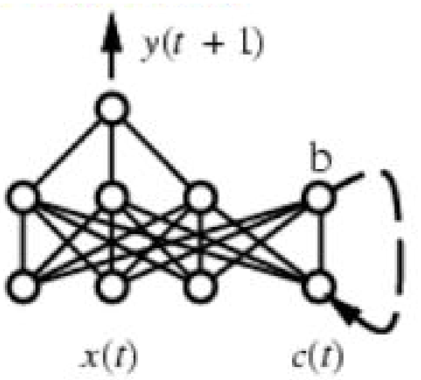
Neural networks for sequential data: recurrent NNs

- In sequential data, history can help predict the future
- Would like to explicitly capture state history in the neural network
  - Add a recurrent state
- In regular RNNs, the recurrent state is overwritten frequently
  - Good if important events happen close together
- But what if there are large delays before important events?
  - Long short-term memory (LSTM) networks!
    - Best known result in connected handwriting recognition
    - Also used for automatic speech recognition



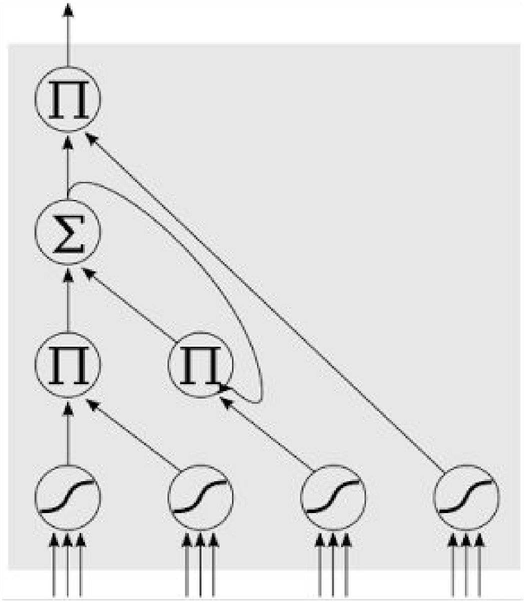
Neural networks for sequential data: recurrent NNs

- In sequential data, history can help predict the future
- Would like to explicitly capture state history in the neural network
  - Add a recurrent state
- In regular RNNs, the recurrent state is overwritten frequently
  - Good if important events happen close together
- But what if there are large delays before important events?
  - Long short-term memory (LSTM) networks!
    - Best known result in connected handwriting recognition
    - Also used for automatic speech recognition
    - In 2016, Google, Apple, Microsoft, Baidu reveal LSTMs as fundamental components in their technologies



LSTMs: remembering values for arbitrary length of time

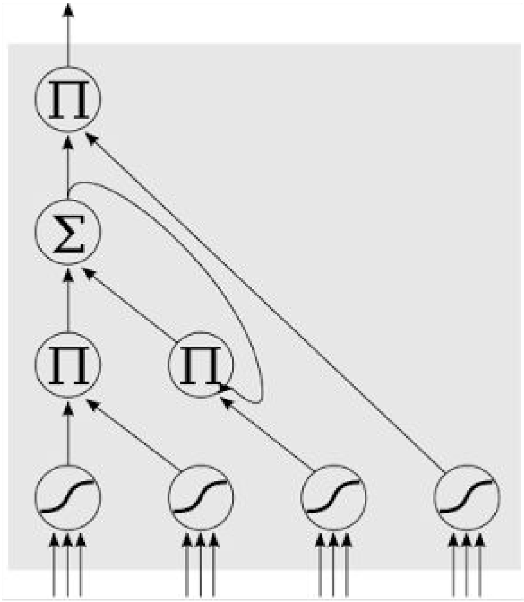
- Contains LSTM blocks



input value   input gate   forget gate   output gate

LSTMs: remembering values for arbitrary length of time

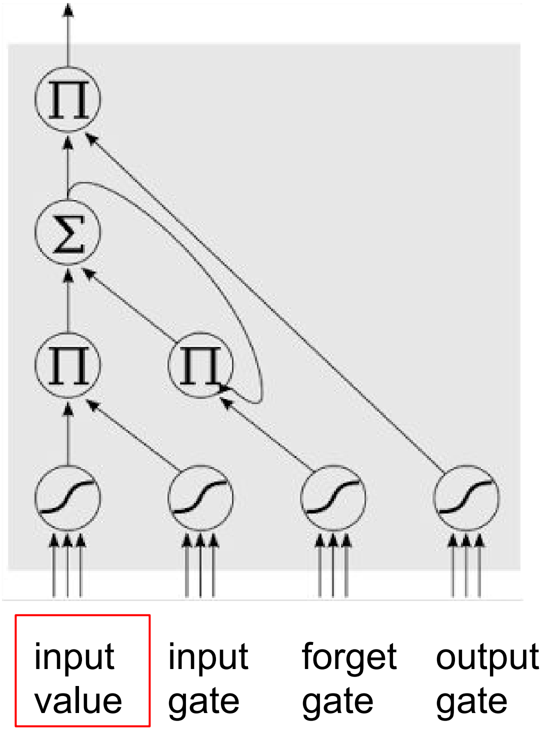
- Contains LSTM blocks
  - each block contains gates that determine when the input is significant enough to remember



input value | input gate | forget gate | output gate

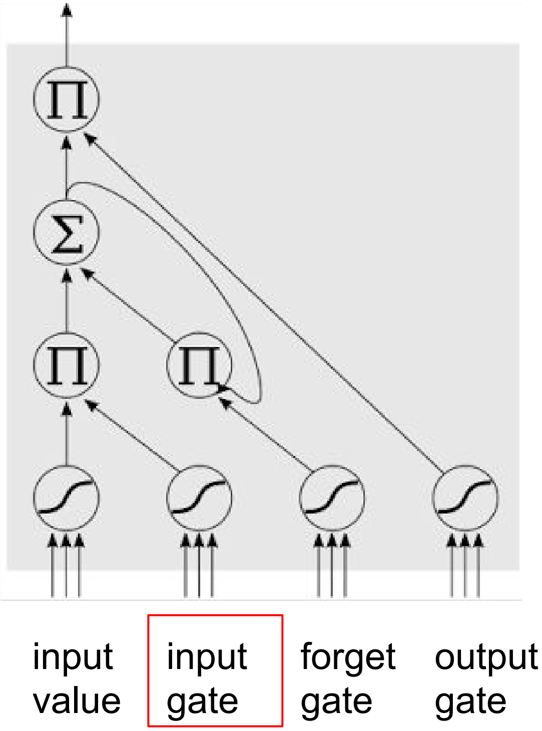
LSTMs: remembering values for arbitrary length of time

- Contains LSTM blocks
  - each block contains gates that determine when the input is significant enough to remember



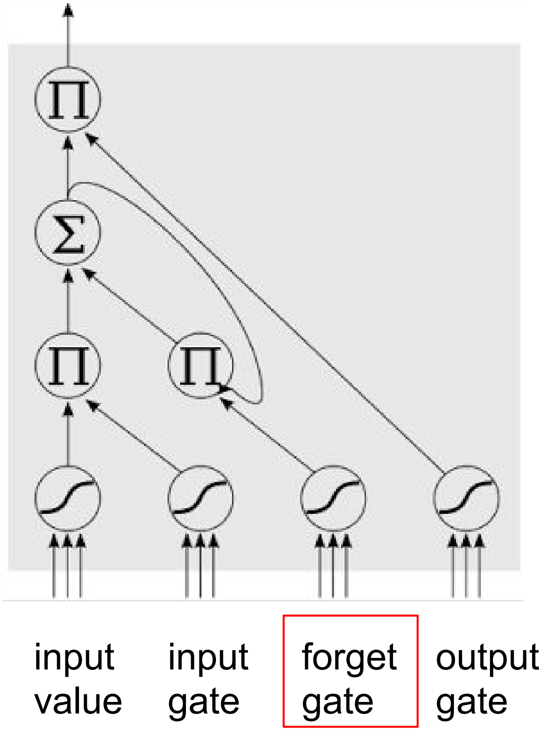
LSTMs: remembering values for arbitrary length of time

- Contains LSTM blocks
    - each block contains gates that determine when the input is significant enough to remember



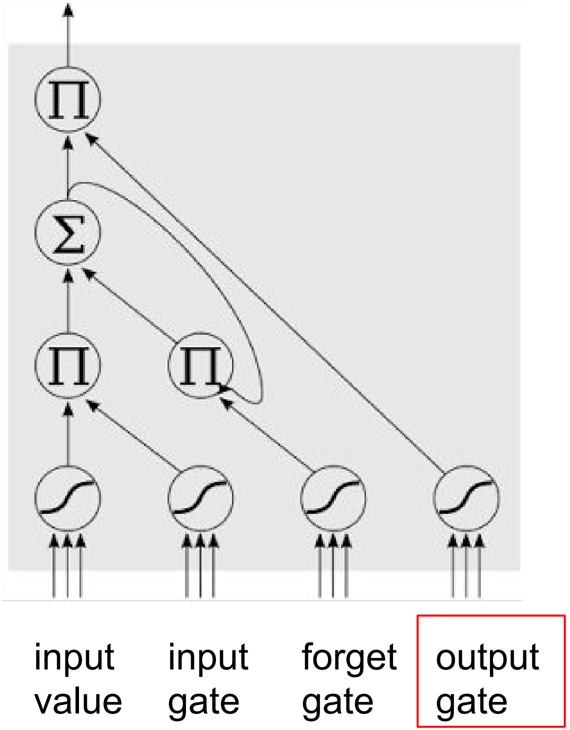
LSTMs: remembering values for arbitrary length of time

- Contains LSTM blocks
  - each block contains gates that determine when the input is significant enough to remember



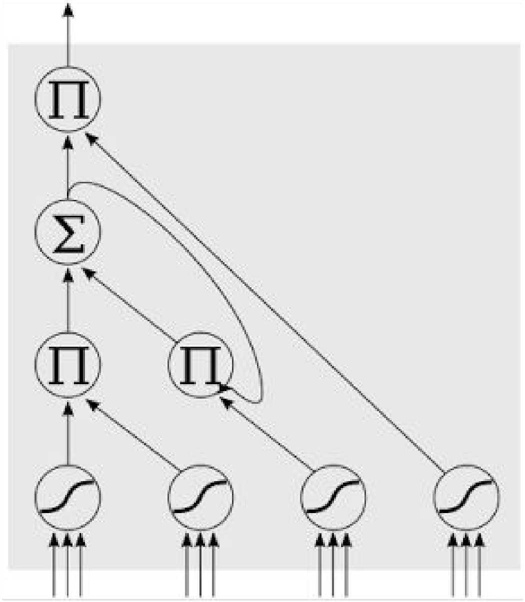
LSTMs: remembering values for arbitrary length of time

- Contains LSTM blocks
  - each block contains gates that determine when the input is significant enough to remember



LSTMs: remembering values for arbitrary length of time

- Contains LSTM blocks
  - each block contains gates that determine when the input is significant enough to remember
- LSTMs, like regular RNNs, can be trained with backpropagation



input value | input gate | forget gate | output gate

3 main problems with backpropagation

- Problem 1: random initialization of weights can lead to bad local minimum

3 main problems with backpropagation

- Problem 1: random initialization of weights can lead to bad local minimum
- Problem 2: learning time does not scale well with number of hidden layers => slow

3 main problems with backpropagation

- Problem 1: random initialization of weights can lead to bad local minimum
- Problem 2: learning time does not scale well with number of hidden layers => slow
- Problem 3: need a lot of data for good estimates of weights when we start from random initializations and to calculate classification error, we need labels => need a lot of labeled data, which is often scarce

Partial solution: deep belief networks (DBN)

- DBNs use a technique called “pre-training” to train each pair of successive layers separately without using labeled data and backpropagation

Partial solution: deep belief networks (DBN)

- DBNs use a technique called “pre-training” to train each pair of successive layers separately without using labeled data and backpropagation
  - uses the resulting pre-trained weights to initialize the DBN weights for the task

Partial solution: deep belief networks (DBN)

- DBNs use a technique called “pre-training” to train each pair of successive layers separately without using labeled data and backpropagation
  - uses the resulting pre-trained weights to initialize the DBN weights for the task
    - avoids random initialization (problem 1 of backpropagation)

Partial solution: deep belief networks (DBN)

- DBNs use a technique called “pre-training” to train each pair of successive layers separately without using labeled data and backpropagation
  - uses the resulting pre-trained weights to initialize the DBN weights for the task
    - avoids random initialization (problem 1 of backpropagation)
  - reduces need for a lot of labeled data (problem 3)

Partial solution: deep belief networks (DBN)

- DBNs use a technique called “pre-training” to train each pair of successive layers separately without using labeled data and backpropagation
  - uses the resulting pre-trained weights to initialize the DBN weights for the task
    - avoids random initialization (problem 1 of backpropagation)
  - reduces need for a lot of labeled data (problem 3)
- Once the DBN is pre-trained, it is fine-tuned with backpropagation for a particular task

Partial solution: deep belief networks (DBN)

- DBNs use a technique called “pre-training” to train each pair of successive layers separately without using labeled data and backpropagation
  - uses the resulting pre-trained weights to initialize the DBN weights for the task
    - avoids random initialization (problem 1 of backpropagation)
  - reduces need for a lot of labeled data (problem 3)
- Once the DBN is pre-trained, it is fine-tuned with backpropagation for a particular task
  - reduces the learning time of backprop (problem 2)

Partial solution: deep belief networks (DBN)

- DBNs use a technique called “pre-training” to train each pair of successive layers separately without using labeled data and backpropagation
  - uses the resulting pre-trained weights to initialize the DBN weights for the task
    - avoids random initialization (problem 1 of backpropagation)
  - reduces need for a lot of labeled data (problem 3)
- Once the DBN is pre-trained, it is fine-tuned with backpropagation for a particular task
  - reduces the learning time of backprop (problem 2)
- So why is it only a partial solution?

Partial solution: deep belief networks (DBN)

- DBNs use a technique called “pre-training” to train each pair of successive layers separately without using labeled data and backpropagation
  - uses the resulting pre-trained weights to initialize the DBN weights for the task
    - avoids random initialization (problem 1 of backpropagation)
  - reduces need for a lot of labeled data (problem 3)
- Once the DBN is pre-trained, it is fine-tuned with backpropagation for a particular task
  - reduces the learning time of backprop (problem 2)
- So why is it only a partial solution?
  - pre-training is done in an unsupervised way, in which the objective is to minimize some reconstruction error

Partial solution: deep belief networks (DBN)

- DBNs use a technique called “pre-training” to train each pair of successive layers separately without using labeled data and backpropagation
  - uses the resulting pre-trained weights to initialize the DBN weights for the task
    - avoids random initialization (problem 1 of backpropagation)
  - reduces need for a lot of labeled data (problem 3)
- Once the DBN is pre-trained, it is fine-tuned with backpropagation for a particular task
  - reduces the learning time of backprop (problem 2)
- So why is it only a partial solution?
  - pre-training is done in an unsupervised way, in which the objective is to minimize some reconstruction error
  - in practice, many times weights learned in this way are not close enough to the ones needed for a classification task, and bad local minima are not avoided

Another partial solution: convolutional neural networks (CNNs)

- Inspired by the structure of the visual system (receptive fields of simple and complex cells)

Another partial solution: convolutional neural networks (CNNs)

- Inspired by the structure of the visual system (receptive fields of simple and complex cells)
- Sparse connections between layers (not fully connected like regular NNs)
    - reduces learning time (problem 2)

Another partial solution: convolutional neural networks (CNNs)

- Inspired by the structure of the visual system (receptive fields of simple and complex cells)
- Sparse connections between layers (not fully connected like regular NNs)
  - reduces learning time (problem 2)
- Often, researchers take an already trained CNN on a related task and fine-tune it for the specific task

Another partial solution: convolutional neural networks (CNNs)

- Inspired by the structure of the visual system (receptive fields of simple and complex cells)
- Sparse connections between layers (not fully connected like regular NNs)
  - reduces learning time (problem 2)
- Often, researchers take an already trained CNN on a related task and fine-tune it for the specific task
  - avoids random initialization of weights (problem 1)

Another partial solution: convolutional neural networks (CNNs)

- Inspired by the structure of the visual system (receptive fields of simple and complex cells)
- Sparse connections between layers (not fully connected like regular NNs)
  - reduces learning time (problem 2)
- Often, researchers take an already trained CNN on a related task and fine-tune it for the specific task
  - avoids random initialization of weights (problem 1)
  - reduces need for labeled data for learning (problem 3)

Another partial solution: convolutional neural networks (CNNs)

- Inspired by the structure of the visual system (receptive fields of simple and complex cells)
- Sparse connections between layers (not fully connected like regular NNs)
  - reduces learning time (problem 2)
- Often, researchers take an already trained CNN on a related task and fine-tune it for the specific task
  - avoids random initialization of weights (problem 1)
  - reduces need for labeled data for learning (problem 3)
- So why only partial solution?

Another partial solution: convolutional neural networks (CNNs)

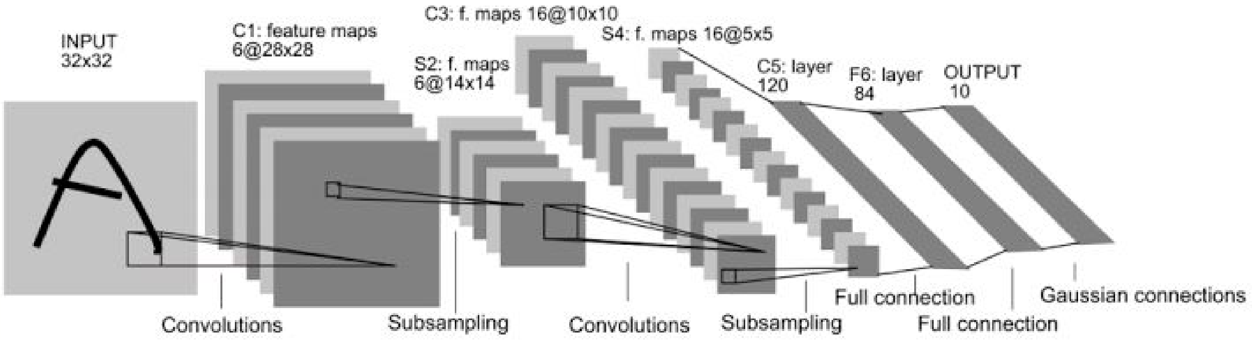
- Inspired by the structure of the visual system (receptive fields of simple and complex cells)
- Sparse connections between layers (not fully connected like regular NNs)
  - reduces learning time (problem 2)
- Often, researchers take an already trained CNN on a related task and fine-tune it for the specific task
  - avoids random initialization of weights (problem 1)
  - reduces need for labeled data for learning (problem 3)
- So why only partial solution?
  - a CNN trained on a related task is not always available => back to problems 1 and 3

Another partial solution: convolutional neural networks (CNNs)

- Inspired by the structure of the visual system (receptive fields of simple and complex cells)
- Sparse connections between layers (not fully connected like regular NNs)
  - reduces learning time (problem 2)
- Often, researchers take an already trained CNN on a related task and fine-tune it for the specific task
  - avoids random initialization of weights (problem 1)
  - reduces need for labeled data for learning (problem 3)

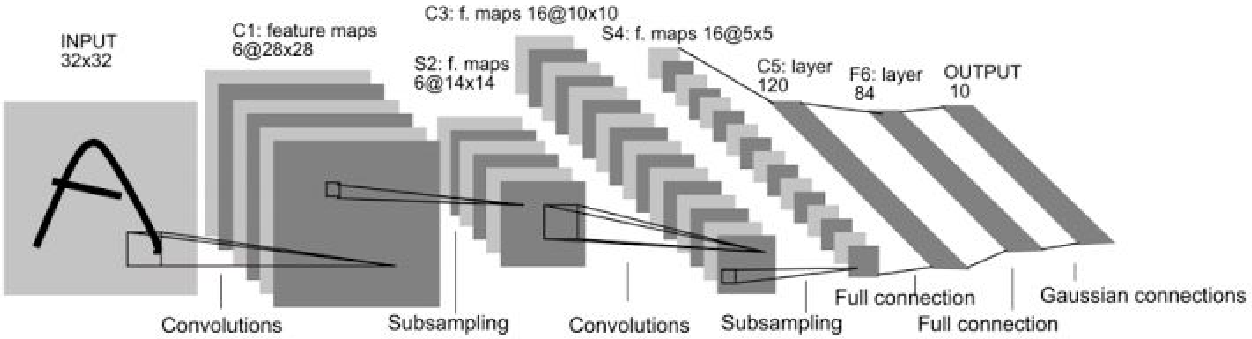
- So why only partial solution?
  - a CNN trained on a related task is not always available => back to problems 1 and 3
  - though, for vision tasks (e.g. classifying images), a good CNN is available

CNN: alternating convolutional and downsampling layers



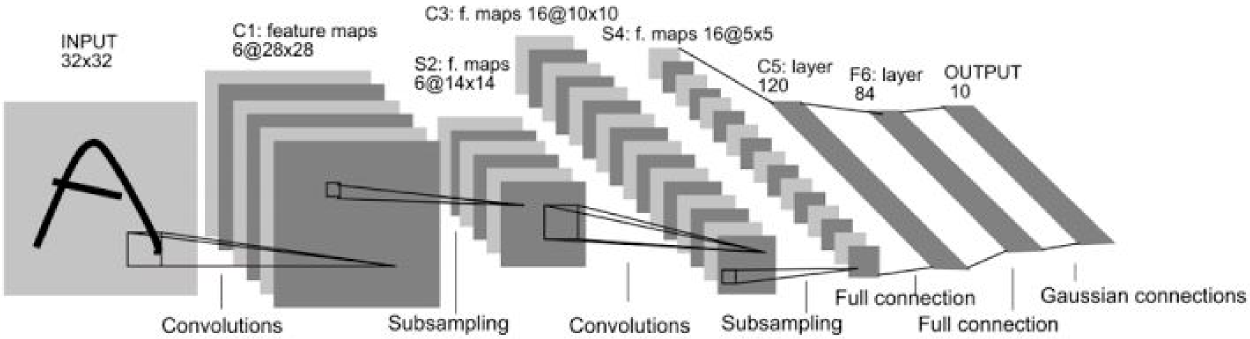
- Restricted connections => makes learning the weights more tractable

CNN: alternating convolutional and downsampling layers



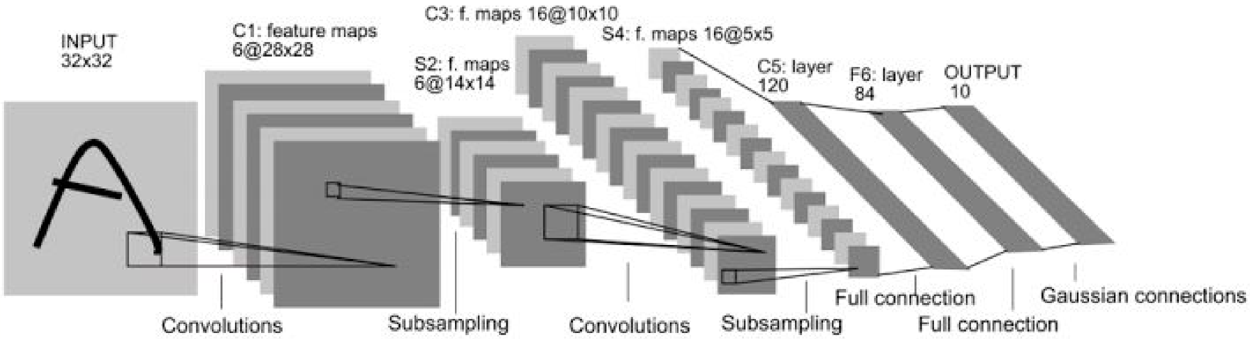
- Restricted connections => makes learning the weights more tractable
- Convolutional layers = each neuron applies some kernel to its receptive field

CNN: alternating convolutional and downsampling layers



- Restricted connections => makes learning the weights more tractable
- Convolutional layers = each neuron applies some kernel to its receptive field
- Subsampling layers = reduce dimensionality, maintain translational invariance

CNN: alternating convolutional and downsampling layers



- ❑ Restricted connections => makes learning the weights more tractable
- ❑ Convolutional layers = each neuron applies some kernel to its receptive field
- ❑ Subsampling layers = reduce dimensionality, maintain translational invariance
- ❑ Shared weights between feature maps reduce number of parameters to learn

Most frequently used CNN is trained on ImageNet

❏ ImageNet is a database of 15M images collected from the web

Most frequently used CNN is trained on ImageNet

- ImageNet is a database of 15M images collected from the web
- 22k categories

Most frequently used CNN is trained on ImageNet

- ImageNet is a database of 15M images collected from the web
- 22k categories
- labels provided by people on MTurk

Most frequently used CNN is trained on ImageNet

- ImageNet is a database of 15M images collected from the web
- 22k categories
- labels provided by people on MTurk
- RGB images (not black and white)

CNN results in ImageNet classification challenge

- 1k categories, 1.2M training images, 50k validation, 150k testing

CNN results in ImageNet classification challenge

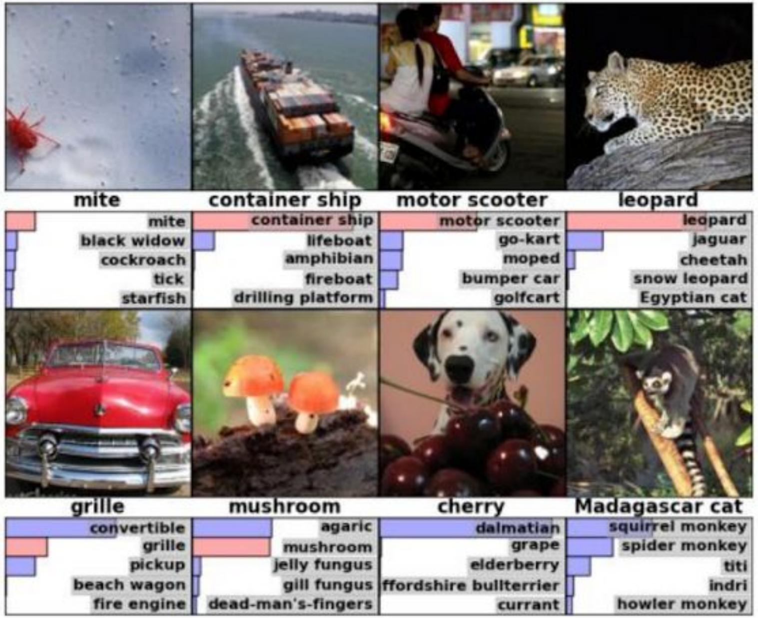
- 1k categories, 1.2M training images, 50k validation, 150k testing
- Classification challenge:
  - Make 1 guess about the label (top-1 error)
  - Make 5 guesses about the label (top-5 error)

CNN results in ImageNet classification challenge

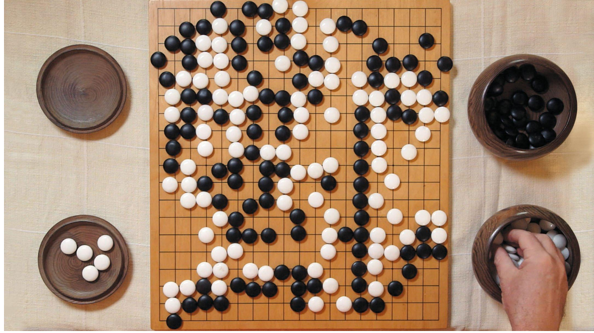
- 1k categories, 1.2M training images, 50k validation, 150k testing
- Classification challenge:
  - Make 1 guess about the label (top-1 error)
  - Make 5 guesses about the label (top-5 error)
- CNN results (won)
  - Top-1 error: 37.5%
  - Top-5 error: 17.0%

CNN results in ImageNet classification challenge

- 1k categories, 1.2M training images, 50k validation, 150k testing
- Classification challenge:
  - Make 1 guess about the label (top-1 error)
  - Make 5 guesses about the label (top-5 error)
- CNN results (won)
  - Top-1 error: 37.5%
  - Top-5 error: 17.0%

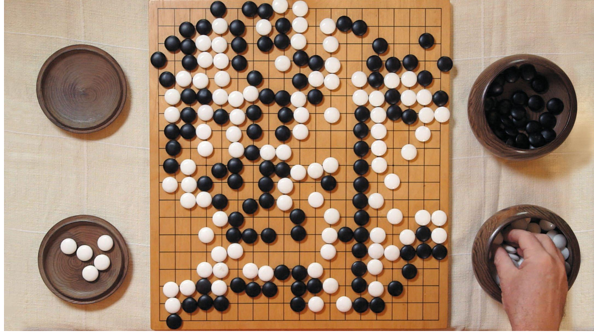


CNN also used in AlphaGo



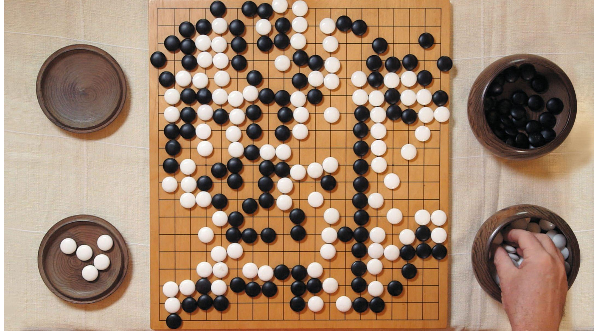
- ❑ First off, why is Go so hard?

CNN also used in AlphaGo



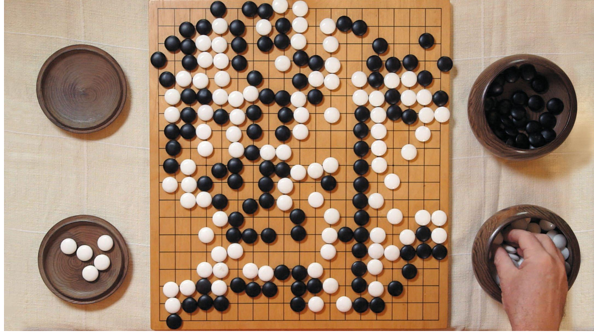
- First off, why is Go so hard?
  - larger board than chess (19x19)

CNN also used in AlphaGo



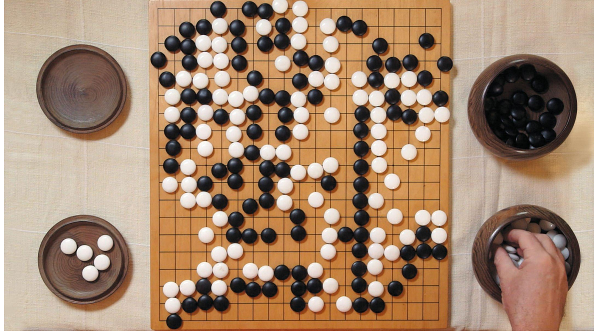
- First off, why is Go so hard?
  - larger board than chess (19x19)
  - many more possible board outcomes than chess

CNN also used in AlphaGo



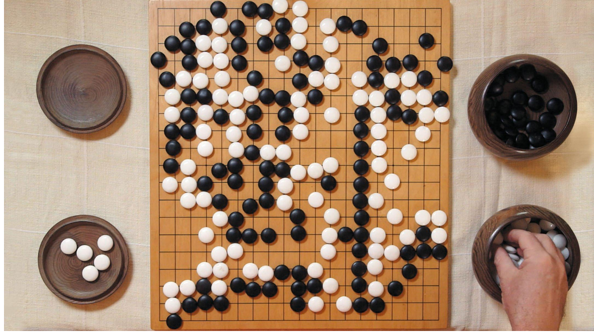
- First off, why is Go so hard?
  - larger board than chess (19x19)
  - many more possible board outcomes than chess
    - each board position has 3 possibilities: empty, black, white

CNN also used in AlphaGo



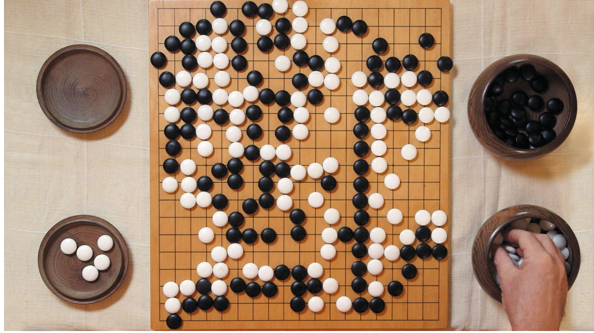
- First off, why is Go so hard?
  - larger board than chess (19x19)
  - many more possible board outcomes than chess
    - each board position has 3 possibilities: empty, black, white
    - an estimated 1.2% of all board positions are legal

CNN also used in AlphaGo



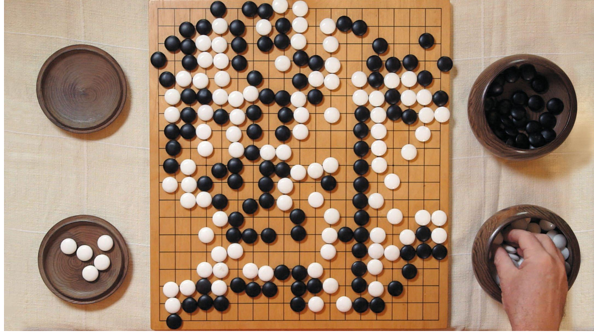
- First off, why is Go so hard?
  - larger board than chess (19x19)
  - many more possible board outcomes than chess
    - each board position has 3 possibilities: empty, black, white
    - an estimated 1.2% of all board positions are legal
    - Total number of possible board positions = 3^{19x19}x1.2% = 2.1x10^{170} > # atoms in universe

CNN also used in AlphaGo



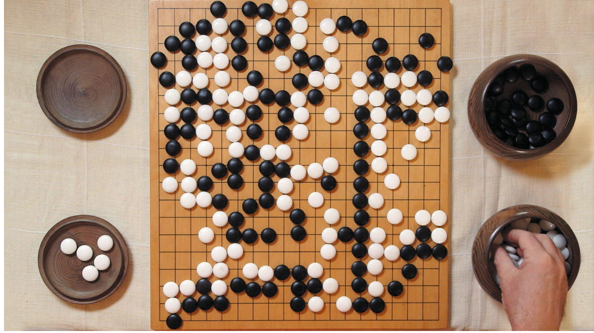
- First off, why is Go so hard?
  - larger board than chess (19x19)
  - many more possible board outcomes than chess
    - each board position has 3 possibilities: empty, black, white
    - an estimated 1.2% of all board positions are legal
    - Total number of possible board positions = 3^{19x19}x1.2% = 2.1x10^{170} > # atoms in universe
- So enumerating all possible board outcomes and scoring them won't work

CNN also used in AlphaGo



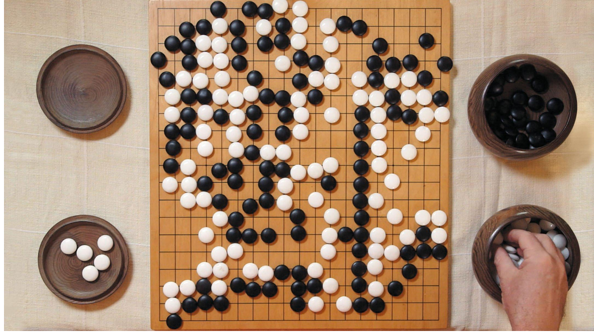
- First off, why is Go so hard?
  - larger board than chess (19x19)
  - many more possible board outcomes than chess
    - each board position has 3 possibilities: empty, black, white
    - an estimated 1.2% of all board positions are legal
    - Total number of possible board positions = 3^{19x19}x1.2% = 2.1×10^{170} > # atoms in universe
- So enumerating all possible board outcomes and scoring them won't work
- Best approach before AlphaGo:

CNN also used in AlphaGo



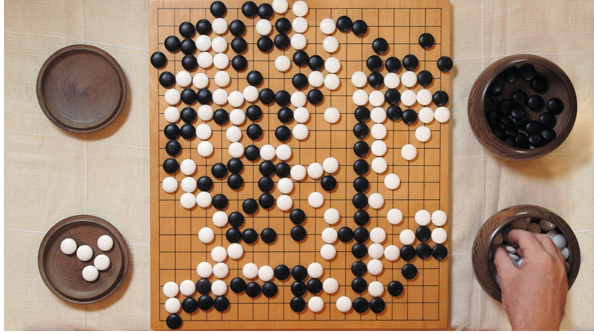
- First off, why is Go so hard?
  - larger board than chess (19x19)
  - many more possible board outcomes than chess
    - each board position has 3 possibilities: empty, black, white
    - an estimated 1.2% of all board positions are legal
    - Total number of possible board positions = 3^{19x19}x1.2% = 2.1x10^{170} > # atoms in universe
- So enumerating all possible board outcomes and scoring them won't work
- Best approach before AlphaGo:
  - Tree search enhanced by policies that were trained to predict human expert's moves

CNN also used in AlphaGo



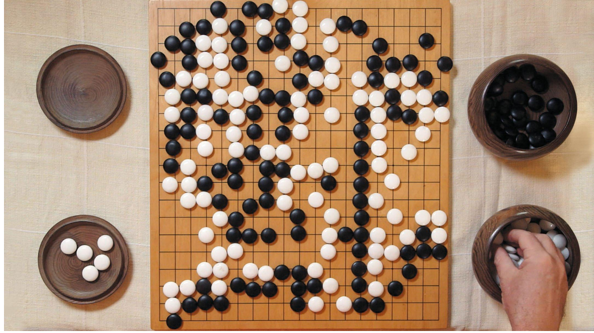
- First off, why is Go so hard?
  - larger board than chess (19x19)
  - many more possible board outcomes than chess
    - each board position has 3 possibilities: empty, black, white
    - an estimated 1.2% of all board positions are legal
    - Total number of possible board positions = 3^{19x19}x1.2% = 2.1x10^{170} > # atoms in universe
- So enumerating all possible board outcomes and scoring them won't work
- Best approach before AlphaGo:
  - Tree search enhanced by policies that were trained to predict human expert's moves
  - "Weak amateur level play"

CNN also used in AlphaGo



- First off, why is Go so hard?
  - larger board than chess (19x19)
  - many more possible board outcomes than chess
    - each board position has 3 possibilities: empty, black, white
    - an estimated 1.2% of all board positions are legal
    - Total number of possible board positions = 3^{19x19}x1.2% = 2.1x10^{170} > # atoms in universe
- So enumerating all possible board outcomes and scoring them won’t work
- Best approach before AlphaGo:
  - Tree search enhanced by policies that were trained to predict human expert’s moves
  - “Weak amateur level play”
- AlphaGo combines tree search with RL executed with policy and value CNN networks

CNN also used in AlphaGo



- First off, why is Go so hard?
  - larger board than chess (19x19)
  - many more possible board outcomes than chess
    - each board position has 3 possibilities: empty, black, white
    - an estimated 1.2% of all board positions are legal
    - Total number of possible board positions = 3^{19x19}x1.2% = $2.1x10^{170}$ > # atoms in universe
- So enumerating all possible board outcomes and scoring them won't work
- Best approach before AlphaGo:
  - Tree search enhanced by policies that were trained to predict human expert's moves
  - "Weak amateur level play"
- AlphaGo combines tree search with RL executed with policy and value CNN networks
  - Won 4 out of 5 games against Lee Sedol, second best player of Go in the world (by # of won

AlphaGo algorithm details

- 3 CNNs - 2 for policy, 1 for value

AlphaGo algorithm details

- 3 CNNs - 2 for policy, 1 for value
- Input to CNNs: board position (19x19 image)

AlphaGo algorithm details

- 3 CNNs - 2 for policy, 1 for value
- Input to CNNs: board position (19x19 image)
- First policy CNN trained to select actions that predict human moves selected in a given state

AlphaGo algorithm details

- 3 CNNs - 2 for policy, 1 for value
- Input to CNNs: board position (19x19 image)
- First policy CNN trained to select actions that predict human moves selected in a given state
  - previous methods used shallow classification methods to do this

AlphaGo algorithm details

- 3 CNNs - 2 for policy, 1 for value
- Input to CNNs: board position (19x19 image)
- First policy CNN trained to select actions that predict human moves selected in a given state
  - previous methods used shallow classification methods to do this
- Second policy CNN trained to select actions that maximize expected future reward (winning) on games between AlphaGo's current policy strategy and a randomly selected previous iteration of the policy strategy

AlphaGo algorithm details

- 3 CNNs - 2 for policy, 1 for value
- Input to CNNs: board position (19x19 image)
- First policy CNN trained to select actions that predict human moves selected in a given state
  - previous methods used shallow classification methods to do this
- Second policy CNN trained to select actions that maximize expected future reward (winning) on games between AlphaGo's current policy strategy and a randomly selected previous iteration of the policy strategy
  - goal: to adjust the policy towards the correct goal of winning games, rather than predictive accuracy

AlphaGo algorithm details

- 3 CNNs - 2 for policy, 1 for value
- Input to CNNs: board position (19x19 image)
- First policy CNN trained to select actions that predict human moves selected in a given state
  - previous methods used shallow classification methods to do this
- Second policy CNN trained to select actions that maximize expected future reward (winning) on games between AlphaGo's current policy strategy and a randomly selected previous iteration of the policy strategy
  - goal: to adjust the policy towards the correct goal of winning games, rather than predictive accuracy
- Value CNN trained to predict the winner of games played by the RL policy network against itself

AlphaGo algorithm details

- 3 CNNs - 2 for policy, 1 for value
- Input to CNNs: board position (19x19 image)
- First policy CNN trained to select actions that predict human moves selected in a given state
  - previous methods used shallow classification methods to do this
- Second policy CNN trained to select actions that maximize expected future reward (winning) on games between AlphaGo's current policy strategy and a randomly selected previous iteration of the policy strategy
  - goal: to adjust the policy towards the correct goal of winning games, rather than predictive accuracy
- Value CNN trained to predict the winner of games played by the RL policy network against itself
- Finally, use the value and policy networks to reduce depth breadth of search

Getting started with deep learning

- A bit of an art

Getting started with deep learning

- A bit of an art
- Several deep learning libraries

Getting started with deep learning

- A bit of an art
- Several deep learning libraries
  - Theano
    - Written in python, compatible with GPUs

Getting started with deep learning

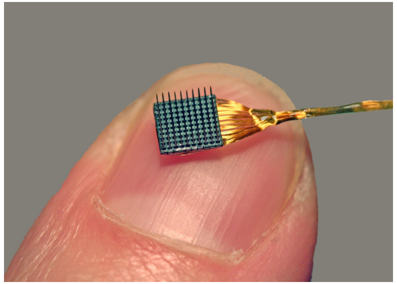
- A bit of an art
- Several deep learning libraries
  - Theano
    - Written in python, compatible with GPUs
  - TensorFlow
    - Similar to Theano but arguably more intuitive
    - https://www.tensorflow.org/

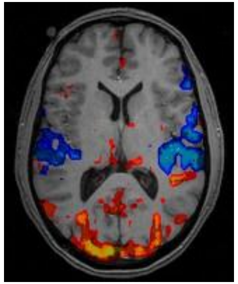
Main takeaways

- Considering the sequential nature of time series data is important
- Ways to account for this time dependence is through models, such as hidden Markov models, Markov decision processes, and long short-term memory neural networks
- We don’t know why exactly deep learning works *yet*, but it’s increasingly more popular, both in industry and academia
    - deep learning course at CMU taught by Ruslan Salakhutdinov in the Fall!

How can ML help neuroscientists?

- Deal with large number of sensors/recording sites

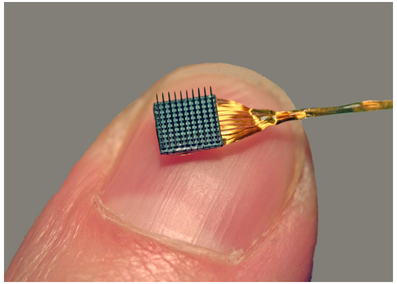


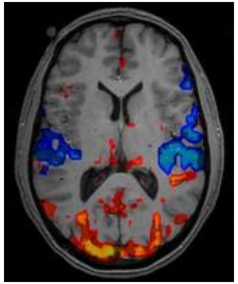


  - investigate high-dimensional representations

How can ML help neuroscientists?

- Deal with large number of sensors/recording sites

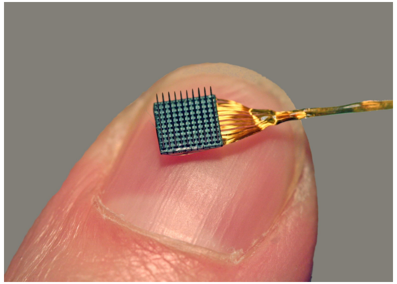
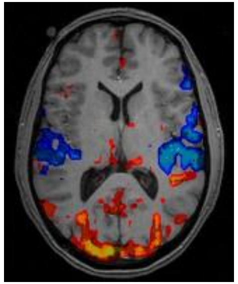




- investigate high-dimensional representations
  - classification (what does this high-dimensional data represent?)

How can ML help neuroscientists?

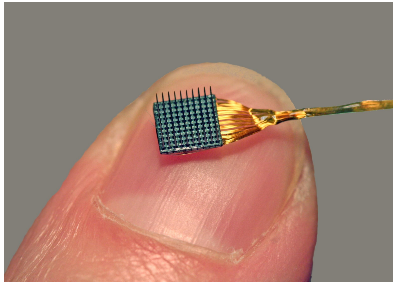
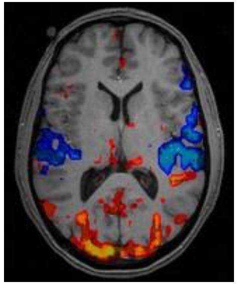
- ❏ Deal with large number of sensors/recording sites

- ❏ investigate high-dimensional representations
  - ❏ classification (what does this high-dimensional data represent?)
  - ❏ regression (how does it represent it? can we predict a different representation?)

How can ML help neuroscientists?

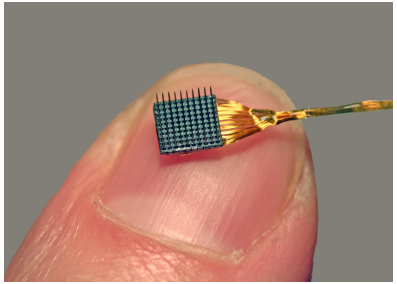
- Deal with large number of sensors/recording sites

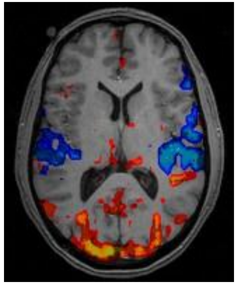
  

  - investigate high-dimensional representations
    - classification (what does this high-dimensional data represent?)
    - regression (how does it represent it? can we predict a different representation?)
    - model selection (what model would best describe this high dimensional data?)

How can ML help neuroscientists?

- ❏ Deal with large number of sensors/recording sites

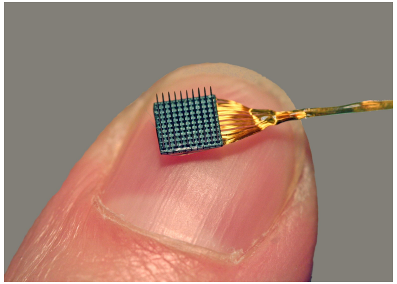
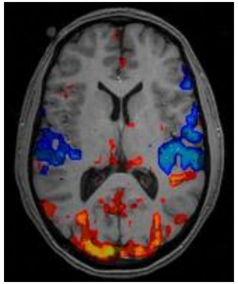




- ❏ investigate high-dimensional representations
  - ❏ classification (what does this high-dimensional data represent?)
  - ❏ regression (how does it represent it? can we predict a different representation?)
  - ❏ model selection (what model would best describe this high dimensional data?)
- ❏ uncover few underlying processes that interact in complex ways

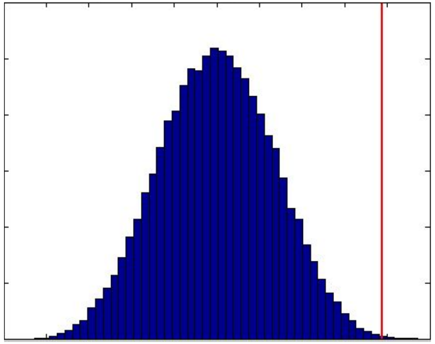
How can ML help neuroscientists?

- Deal with large number of sensors/recording sites

- investigate high-dimensional representations
  - classification (what does this high-dimensional data represent?)
  - regression (how does it represent it? can we predict a different representation?)
  - model selection (what model would best describe this high dimensional data?)
- uncover few underlying processes that interact in complex ways
  - dimensionality reduction techniques

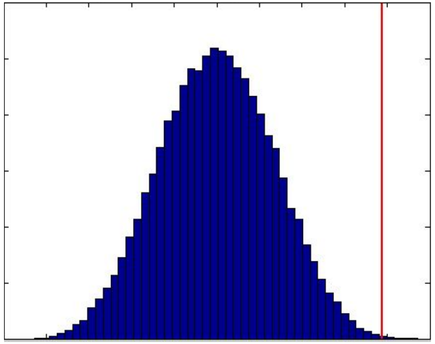
How can ML help neuroscientists?



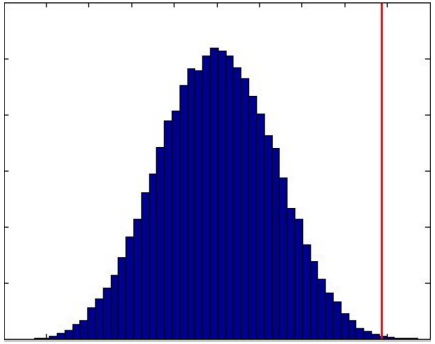
- Evaluate results

How can ML help neuroscientists?

- Evaluate results
  - cross validation (how generalizable are our results?)

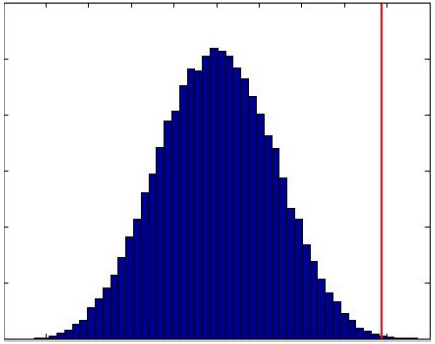


How can ML help neuroscientists?



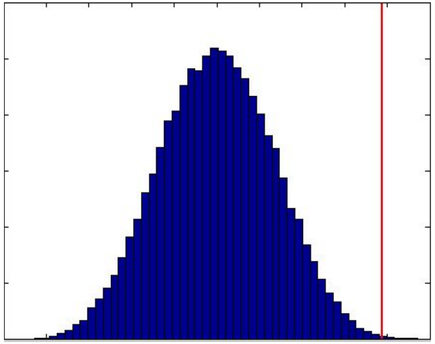
- Evaluate results
  - cross validation (how generalizable are our results?)
  - nearly assumption-free significance testing (are the results different from chance?)

How can ML help neuroscientists?

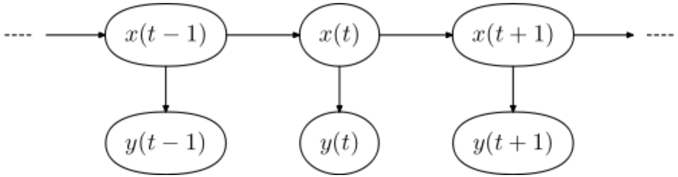


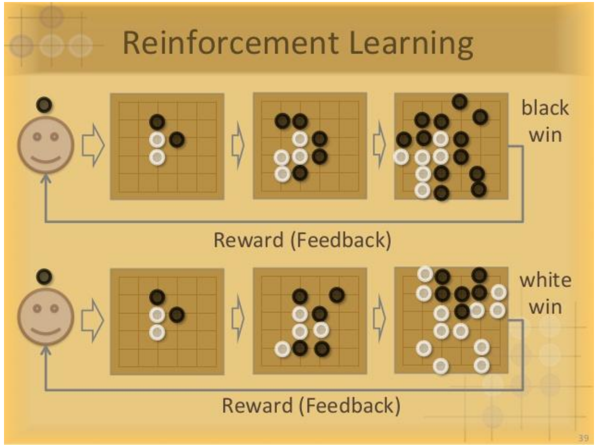
- Evaluate results
  - cross validation (how generalizable are our results?)
  - nearly assumption-free significance testing (are the results different from chance?)
- Complex data-driven hypotheses of brain processing

How can ML help neuroscientists?



- ❑ Evaluate results
  - ❑ cross validation (how generalizable are our results?)
  - ❑ nearly assumption-free significance testing (are the results different from chance?)
- ❑ Complex data-driven hypotheses of brain processing
  - ❑ advanced topics: latent variable models, reinforcement learning, deep learning





Thank you for your attention!

Please fill out the anonymous evaluation sheet -- your feedback is very important!

Looking forward to questions or follow-ups:

mariya@cmu.edu